An Obstacle Avoidance Algorithm for a Mobile Robot Based upon the Potential Field Method

T. Wilschut

ADVISORS:
Dr. ir. L.F.P Etman (Manufacturing Networks group)
Dr. ir. G.J.L. Naus (Systems and Control Technology group)

Eindhoven, November 2011

Nr: 2011-420667
Abstract

The main problem considered in this paper is as follows; a robot, at position $\mathbf{p}_R$, needs to navigate in a finite space $C$ around dynamic or stationary obstacles to a target at position $\mathbf{p}_{\text{tar}}$. During navigation problems like finite acceleration and velocity of the robot, local minima and space boundaries should be accounted for.

Prior to this paper an algorithm based upon Newton-Direction was created by Mouton (2011). Newton direction motion planning uses an artificial potential field $U$ to model the space around the robot. In $U$ the target is represented by a global minimum and obstacles by local maxima. By locally differentiating this potential field the field gradient can be obtained. The field gradient is modified using the Inverse Hessian function. The resulting vector is considered as the most promising direction of motion. Mouton used the placement of virtual obstacles to accommodate for local minima and space boundaries.

In this paper the methods used by Mouton are compared to methods published in literature to solve similar problems. Special emphasis is put on the potential field method. Robot motion planning based upon the potential field method has strong similarities to Newton direction navigation, only the field gradient, considered as the most promising direction of motion, is not modified with the inverse Hessian function but the field gradient itself is considered as the most promising direction of motion.

From literature some solutions will be chosen to be implemented into the algorithm. A modified potential field is introduced; in which the potential is not only a function of the position of the robot $\mathbf{p}_R$ but also of the velocity $\mathbf{v}_R$ of the robot; an ‘escape force’ is used to avoid local minima and the dynamical model of the robot is used to enable the algorithm to trace the direction of the field gradient.

Finally, a simulation study will be done to compare both algorithms. It will be shown that the new developed algorithm has a consistently shorter task completion time; generates a smoother velocity profile; can safely reach a target near an obstacle; can easily avoid a head on collision at maximum velocity; can trace a dynamic target which previously was not possible and that the target approach phase can be controlled to be undamped, damped or over-damped due to the usage of the velocity information in the formulation of the potential field $U$. 
Notations

\( \vec{p}_R \), position vector of robot
\( \vec{p}_{\text{tar}} \), position vector of target
\( \vec{p}_{\text{obst},i} \), position vector of obstacle \( i \)
\( \vec{p}_{\text{RT}} \), difference vector between robot and target
\( \vec{p}_{\text{ROI}} \), difference vector between robot and obstacle \( i \)
\( \vec{v}_{\text{RT}} \), relative velocity between robot and target
\( \vec{v}_{\text{ROI}} \), relative velocity between robot and obstacle \( i \)
\( \vec{n}_{\text{RT}} \), unit vector pointing from robot towards target
\( \vec{n}_{\text{ROI}} \), unit vector pointing from robot towards obstacle \( i \)
\( \vec{n}_{\text{RT} \perp} \), unit vector perpendicular to \( \vec{n}_{\text{RT}} \)
\( \vec{n}_{\text{ROI} \perp} \), unit vector perpendicular to \( \vec{n}_{\text{ROI}} \)
\( \rho_T \), Euclidean distance between robot and target
\( \rho_{s,i} \), Euclidean distance between robot and obstacle \( i \)
\( \rho_R \), Radius of robot
\( \rho_m \), distance travelled while reducing \( v_{\text{ROI}} \) to zero
\( \rho_{oi} \), influence zone of obstacle \( i \)
\( \vec{F} \), artificial force vector
\( \hat{\vec{F}}_p \), position component of artificial force
\( \hat{\vec{F}}_v \), velocity component of artificial force
\( \hat{\vec{F}}_{\text{att}} \), attractive artificial force vector
\( \hat{\vec{F}}_{\text{att},p} \), position component of attractive artificial force
\( \hat{\vec{F}}_{\text{att},v} \), velocity component of attractive artificial force
\( \hat{\vec{F}}_{\text{rep}} \), repulsive artificial force vector
\( \hat{\vec{F}}_{\text{rep},p} \), position component of repulsive artificial force
\( \hat{\vec{F}}_{\text{rep},v} \), velocity component of repulsive artificial force
\( \hat{\vec{F}}_e \), escape force vector
\( \vec{U} \), potential field
\( \vec{U}_{\text{att}} \), attractive potential field
\( \vec{U}_{\text{rep}} \), repulsive potential field
\( \alpha_p \), positive scalar constant
\( \alpha_v \), positive scalar constant
\( \alpha_e \), positive scalar constant
\( m \), positive scalar constant
\( n \), positive scalar constant
\( k \), positive scalar constant
\( \varepsilon \), arbitrarily small constant to prevent singularities in algorithm
\( \vec{V}_{\text{max}} \), maximum velocity of robot
\( \vec{V}_{\text{cur}} \), current robot velocity vector
\( \vec{V}_{\text{des}} \), desired robot velocity vector
\( d\vec{V} \), error between desired and current velocity
\( \alpha_{\text{max}} \), maximum acceleration of robot
\( \vec{a}_{\text{app}} \), applied acceleration
\( \tau_0 \), maximum available control force
\( \tau \), applied control force
\( M \), Inertia matrix
\( m_R \), mass of robot
\( N \), body forces
\( \xi \), damping ratio
\( H \), Hessian function

Notations used by Mouton

\( f_{\text{target}} = \vec{U}_{\text{att}} \), target function (attractive potential)
\( f_{\text{barrier},i} = \vec{U}_{\text{rep},i} \), barrier function (repulsive potential of obstacle \( i \))
\( f_{\text{total}} = \vec{U} \), sum of target and barrier functions (Potential field)
\( P_{f,i} \), penalty function
\( W_t \), target weigh factor
\( \vec{g}_{f} = \nabla f_{\text{Target}} \), field gradient
\( \vec{d} \), Newton direction vector
\( \alpha \), critical collision angle
\( d\vec{V}_{\text{max}} \), maximum velocity change in one time step
## Contents

1. Introduction - 4 -

2. General concept of the potential field Method - 6 -
   2.1 Definitions - 6 -

3. Methods used in Mouton - 9 -
   3.1 Solutions for mathematical problems - 10 -
   3.2 Solutions for physical problems - 11 -
   3.3 Dynamic obstacles - 13 -

4. Extensions upon the potential field method - 14 -
   4.1 Oscillation problems - 16 -
   4.2 Local and global minima problems - 17 -
   4.3 Modeling of Field boundaries and non-circular obstacles - 18 -
   4.4 Dynamical model for an holonomic robot with physical limitations - 19 -

5. A comparison between methods - 21 -
   5.1 Local and Global minima - 22 -
   5.2 Desired direction of motion computation - 22 -
   5.3 Physical limitations and field boundaries - 23 -

6. Implementation concept - 25 -
   6.1 The desired direction of motion - 25 -
   6.2 Construction and application of the inverse Hessian function $H^{-1}$ - 26 -
   6.3 Addition of the escape force $F_e$ - 27 -
   6.4 Gradient tracing - 28 -

7. Simulation results: System behaviour - 29 -
   7.1 Desired behaviour - 29 -
   7.2 Obstacle avoidance phase - 29 -
      7.2.1 Conclusion - 32 -
7.3 Target approach phase
  7.3.1 Conclusion
7.4 Influence of the inverse Hessian function
  7.4.1 Conclusion
7.5 Escaping local minima
7.6 Dynamical obstacles
7.7 Dynamical Target
7.8 Space Boundaries

8. Simulation results: comparison
  8.1 One obstacle
  8.2 Local minima
    8.2.1 Two Obstacles
    8.2.2 Three obstacles
  8.3 Dynamical Environment
  8.4 Target within influence zone of robot
  8.5 Conclusion

9. Conclusion
  9.1 Recommendations for future work

References

Appendices
  A.1: The Hessian function
  A.2: The influence of $m, n, \alpha_p, \text{ and } \alpha_v$ on the potential function.
  A.3: The new repulsive Force
  A.4: Hessian components
  A.5: Data structures and code tree
  A.6: M-files
1. Introduction

During the last decade the interest in autonomous operating robots has grown. Instead of pre-defining all possible actions in all possible situations into the robots algorithm, the robot is programmed to have a certain level of artificial intelligence which allows the robot to choose the most desirable action in an arbitrary situation by itself. An important aspect of this artificial intelligence is robot motion planning. In most real life situations there is no prior model of the environment around the robot, which implies that the robot cannot follow a pre-defined path but has to create one during motion execution.

Figure 1.1 represents the main problem considered in this paper; a robot, at position $\vec{p}_R$, needs to navigate in a finite space $C$, spanned by the orthonormal vectors $x$ and $y$, around dynamic or stationary obstacles to a target at position $\vec{p}_{tar}$. Special emphasis is put on the Potential Field Method in order to find a solution for the problem. This method was originally designed as a fast on-line collision avoidance approach. (Khatib, (1980)).

The basic concept behind the potential field method is as follows: The robot is attracted by the target and repelled by obstacles. Using this information an artificial potential field $U$ is used to model the environment around the robot. Generally, the target position is represented by a local minimum and obstacles by local maxima. Often the shape of the potential field $U$ does not depend on the configuration of obstacles beyond a predefined distance from the robot; this is why the potential field method is often referred to as a ‘local’ method.

By locally differentiating the potential field $U$, an artificial force $\vec{F}(\vec{p}_R)$ is obtained. The direction of $\vec{F}(\vec{p}_R)$, is considered as the most promising direction of motion. Motion planning is done iteratively; at each step, the artificial force $\vec{F}(\vec{p}_R) = -\nabla U(\vec{p}_R)$ resulting from the artificial potential field at the current position of the robot $\vec{p}_R$ is calculated. Subsequently, $\vec{F}(\vec{p}_R)$ is converted into an actual movement of the robot. Figure 1.2 illustrates a robot travelling at a velocity $\vec{v}_R$ and the artificial force $\vec{F}(\vec{p}_R)$; two approaches can be used to convert $\vec{F}(\vec{p}_R)$ into an actual movement of the robot.

Figure 1.2: a) $\vec{F}_{app}$ is applied in the direction of $\vec{F}$. b) $\vec{F}_{app}$ is applied to converge to the direction of $\vec{F}$
Method (a) applies a force $\hat{F}_{app}$ (acceleration) parallel to $\vec{F}$ which results in slow convergence towards the desired direction of motion. Method (b) applies a force $\hat{F}_{app}$ in the direction of the difference vector between $\vec{v}_R$ and $\vec{F}$ which ensures fast convergence to the desired direction of motion.

During the creation of the potential field method focus was put on real-time efficiency, rather than on assignment completion assurance (*Latombe (1991)*). Since the potential field method in essence works as a fastest descent optimization procedure, it could steer toward a local minimum of the potential function other than the target position. Moreover, it may not be able to always find a free path even if one exist.

Prior to this paper *van de Bulk (2010)* developed an algorithm which used, similar to the potential field method, a potential field from which the derivative is taken in order to determine the desired direction of motion. In addition to the potential field method; the desired direction of motion is modified by multiplying it with the inverse of the Hessian function. The Hessian function contains all second order partial derivatives of the potential field. The resulting direction is called the Newton Direction. Subsequently *van Heur (2010), Vermeulen (2011), and Mouton (2011)* expanded upon this algorithm which resulted in the algorithm used as a start of this paper. Since Mouton was the last to adapt the algorithm, it will be referred to as Mouton.

The current algorithm is able to cope with local minima problems, dynamic obstacles and field boundaries. Mouton uses virtual obstacle placement to push the robot away from local minima, i.e. if the robot is stuck in a local minima a virtual obstacles will be placed behind the robot to push it onwards. Dynamic obstacles are contributed for in a similar fashion; the collision point is calculated and a virtual obstacle is placed at that position, driving the robot away from the collision point. Parameter settings are used to determine which obstacles are relevant and which are not.

![Virtual obstacle placement](image)

*Figure 1.2: Placement of a virtual obstacle to prevent the robot from colliding with another dynamic obstacle.*

The Mouton algorithm is not complete, i.e. it cannot always reach the target or avoid collisions even when a free path is present. The goal of this paper is to compare the methods used in Mouton with research on potential field methods published in literature to find out what the similarities and differences are. Secondly the goal is to find out where improvements can be made in future implementation. In order do so, the algorithm originally developed by Mouton is adapted, reflecting methods published in literature based upon the potential field method. Simulations are run to compare the new algorithm with the original developed algorithm by Mouton.

The report starts with a literature study. Chapter two explains the general concept of the potential field method and the notation used throughout the paper. Chapter three gives insight in the methods used in Mouton. Chapter four gives variations and expansion on the general potential field method explained in chapter two. Subsequently, a comparison will be made between Mouton and literature in chapter five. From this literature study a few methods are selected which are suitable for implementation and which improve the original algorithm. In Chapter six the implementation concept will be explained.

The second part of this paper consist of the implementation and testing of the methods selected in Chapter six. Simulations have been done to determine the system behavior while the robot is avoiding obstacles and when the robot approaches the target. These results are compared to the desired behavior: generation of a smooth and short obstacle avoidance trajectory with little to non oscillations. In the end, the performance of the newly developed algorithm will be compared to the prior developed algorithm of Mouton.
2. General concept of the potential field method

Motion planning algorithms which make use of the potential field method are all based upon the following idea; the robot is attracted towards the target position and is repelled by obstacles. In order to realize this, the robot is treated as a point in space which is influenced by an artificial potential field $U$. The local variations within $U$ represent the configuration of the robot, the target and the obstacles in space, i.e. the target position is represented by a global minima and the obstacles are denoted by local maxima. Generally, the potential field is defined as the sum of an attractive potential $U_{\text{att}}$, which pulls the robot towards the target, and a repulsive potential $U_{\text{rep}}$, which pushes the robot away from obstacles.

2.1 Definitions

In this report, a robot is considered which moves freely in two dimensional space, $C = R^2$. Throughout the report the definitions as defined in Figure 2.1 are used to describe the space configuration. The positions of the robot, the target and the obstacles are denoted with $\vec{p}_R$, $\vec{p}_{\text{tar}}$ and $\vec{p}_{\text{obst},i}$ respectively. The difference vector between robot and target is denoted with $\vec{p}_{\text{RT}}$, the difference vectors between robot and obstacles are denoted with $\vec{p}_{\text{ROI}}$. The constant $\rho_{\text{oi}}$ indicates the influence region of an obstacle, i.e. for $\|\vec{p}_{\text{ROI}}\| < \rho_{\text{oi}}$ the obstacle is considered relevant and influences the path of the robot.

![Figure 2.1: Space configuration at time t.](image)

The potential field function $U(\vec{p}_R)$ is the sum of an attractive potential field $U_{\text{att}}(\vec{p}_R)$ and a repulsive potential field $U_{\text{rep}}(\vec{p}_R)$. Path construction is done iteratively. At each step, the derivative of the potential field function is calculated which results in the so called artificial force $\vec{F}$ which is given by equation (2). The direction of the artificial force vector $\vec{F}$ is considered as the most desirable direction of motion. Similarly to the potential field function, the artificial force $\vec{F}$ can also be separated into an attractive part $\vec{F}_{\text{att}}$ and a repulsive part $\vec{F}_{\text{rep}}$

$$U(\vec{p}_R) = U_{\text{att}}(\vec{p}_R) + U_{\text{rep}}(\vec{p}_R) \quad (1)$$

$$\vec{F}(\vec{p}_R) = -\nabla U(\vec{p}_R) \quad (2)$$

The attractive potential function is defined as in equation (3), where $\alpha_p$ is a positive scaling factor and $m$ a positive constant.

$$U_{\text{att}}(\vec{p}_R) = \alpha_p \rho_{\text{RT}}^m \quad (3)$$

$\rho_{\text{RT}}(\vec{p}_R)$ defines the Euclidean distance between robot and target given by:

$$\rho_{\text{RT}} = \|\vec{p}_{\text{RT}}\| = \|\vec{p}_{\text{tar}} - \vec{p}_R\| = (\vec{p}_{\text{tar}} - \vec{p}_R) \cdot \hat{\vec{n}}_{\text{RT}} \quad (4)$$
Where $\vec{n}_{RT} = \frac{\vec{p}_{RT}}{\|\vec{p}_{RT}\|}$, is a unit vector pointing form the robot to the target.

Defining the attractive potential as in (3) results in a function which is null in the target position, i.e. $\rho_T = 0$, and approaches infinity when $\rho_T \to \infty$. For different values of $m$ different shapes of the attractive potential field can be realized. For example; $m = 1$ results in a conic well and $m = 2$ in a parabolic well.

Since the function $\rho_T(\vec{p}_R)$ is differentiable in $C$ it is also differentiable for every configuration $\vec{p}_R$, i.e. for all possible positions of the robot, the target and the obstacles in $C$. The resulting attractive force is:

$$\vec{F}_{att}(\vec{p}_R) = -\nabla_{\vec{p}_R} U_{att}(\vec{p}_R)$$

$$= -\gamma_{\rho R} \alpha \rho_{T}^{m-1} \nabla_{\rho R} \rho_T$$

$$= -m \alpha \rho_{T}^{m-1} \nabla_{\rho R} \rho_T$$

(5)

Substituting equation (4) and the relationship for $\vec{n}_{RT}$ results in (6)

$$= -m \alpha \rho_{T}^{m-1} \nabla_{\rho R} (\vec{p}_{tar} - \vec{p}_R) \cdot \vec{n}_{RT}$$

$$= -m \alpha \rho_{T}^{m-1} (-\vec{n}_{RT})$$

$$= -m \alpha \rho_{T}^{m-1} (-\vec{n}_{RT})$$

$$= m \alpha \rho_{T}^{m-2} \vec{p}_{RT}$$

(6)

It is the presence of the derivative of $\rho_T$ which converts the scalar function $U_{att}$ into a vector function $\vec{F}_{att}$ which points from the robot to the target.

To assure the robot does not collide with an obstacle the repulsive function needs to create a barrier which cannot be crossed. Secondly it is preferred that an obstacle does not influence the robot trajectory when it is outside the influence zone $\rho_{oi}$. These constraints can be satisfied by defining the repulsive function as in equation (7).

$$U_{rep,i}(\vec{p}_R) = \begin{cases} 0 & \text{if } \rho_{s,i} > \rho_{oi} \\ \eta \left( \frac{1}{\rho_{s,i}(\vec{p}_R)} - \frac{1}{\rho_{oi}} \right) & \text{if } \rho_{s,i} \leq \rho_{oi} \end{cases}$$

(7)

Where $\eta$ is a positive scaling factor and $\rho_{s,i}$ the Euclidean distance between the robot and obstacle $i$.

$$\rho_{s,i} = \|\vec{p}_{ROI} - \vec{p}_{obst,i} - \vec{p}_R\| = \left(\vec{p}_{obst,i} - \vec{p}_R\right) \cdot \vec{n}_{ROI}$$

(8)

With $\vec{n}_{ROI} = \frac{\vec{p}_{ROI}}{\|\vec{p}_{ROI}\|}$ a unit vector pointing from the robot to obstacle $i$. Defining $U_{rep}$ in such a way assures $U_{rep} = 0$ when $\rho_{s,i} \geq \rho_{oi}$ and $U_{rep} \to \infty$ when $\rho_{s,i} \to 0$. Similar to the artificial attractive force the artificial repulsive force is defined as the negative gradient of the repulsive potential field.

$$\vec{F}_{rep,i}(\vec{p}_R) = -\nabla_{\vec{p}_R} U_{rep,i}(\vec{p}_R)$$

$$= -\nabla_{\vec{p}_R} \eta \left( \frac{1}{\rho_{s,i}} - \frac{1}{\rho_{oi}} \right)$$
\[ F_{\text{rep},i}(\vec{p}_R) = \frac{1}{\rho_{z,i}(\vec{p}_R)} \nabla_{\vec{p}_R} \rho_{z,i} \]
\[ = -\eta \left( \frac{1}{\rho_{z,i}(\vec{p}_R)} \right) \vec{n}_{ROI} \]  

(9)

\( F_{\text{rep},i}(\vec{p}_R) \) is a vector pointing in the opposite direction of \( \vec{n}_{ROI} \), pushing the robot away from obstacle \( i \). The total repulsive force can be defined as the sum of all repulsive forces.

\[ F_{\text{rep}}(\vec{p}_R) = \sum_{i=1}^{n_{\text{obst}}} F_{\text{rep},i}(\vec{p}_R) \]  

(10)

With \( n_{\text{obst}} \) the number of obstacles. Figure 2.2 shows the direction of the resultant direction of the artificial force \( \vec{F}(\vec{p}_R) \) given by equation (11)

\[ \vec{F}(\vec{p}_R) = \vec{F}_{\text{att}}(\vec{p}_R) + \vec{F}_{\text{rep}}(\vec{p}_R) \]  

(11)

*Figure 2.2: Resultant direction of the artificial force \( \vec{F}(\vec{p}_R) \) due to the attractive and repulsive forces*
3. Methods used in Mouton

The basic idea behind the method used in Mouton is similar to the potential field method: the robot is attracted to the goal and repelled by the obstacles. The environment around the robot is modeled as a three-dimensional world described by the sum of a target function (attractive potential), which represents the target, and barrier functions (repulsive potential), which represent obstacles. For the ease of computation, the physical size of the robot is compensated for in the obstacle size. The robot itself is modeled as a single point. The target function is given by equation (12): $f_{\text{target}}$ is a quadratic function with its minimum at the target location. By doing so the field is defined at all possible locations and the robot will move directly towards the target, if not hindered by obstacles.

$$f_{\text{target}} = \|\vec{p}_R - \vec{p}_\text{tar}\|^2$$  \hspace{1cm} (12)

In equation (12), $f_{\text{target}}$ represents the attractive potential field where $\vec{p}_R$, $\vec{p}_\text{tar}$ are the robot and the target locations respectively. The obstacles are accounted for using equation (13); $f_{\text{barrier},i}$, called the barrier function. It creates $n_{\text{obst}}$ infinitely high pillars at the obstacle locations, where $n_{\text{obst}}$ is the number of obstacles. All obstacles are approximated by circles. The radius $r_{\text{obst},i}$ of the obstacles is increased with the radius of the robot, $\rho_R = 0.25 \text{ m}$.

$$f_{\text{barrier},i} = \left(\frac{1}{r_{\text{obst},i}}\frac{1}{\|\vec{p}_{\text{obst},i} - \vec{p}_R\|}\right) , i = 1, \ldots, n_{\text{obst}}$$  \hspace{1cm} (13)

Since not all obstacles are relevant during navigation each is given a penalty factor $P_{f,i}$. In order to determine which obstacles are significant and which are not, three criteria are established.

1. Obstruction, i.e. the obstacle is in between the robot and the target. Mathematically this is the case when the dot product between the vector pointing from the robot to the obstacle and the vector pointing from the robot to the target is smaller than a critical ‘collision angle $\alpha$’. Secondly, the obstacle has to be closer to the robot than the target.

$$\frac{\vec{p}_{\text{obst},i} - \vec{p}_R}{\|\vec{p}_{\text{obst},i} - \vec{p}_R\|} \cdot \frac{\vec{p}_\text{tar} - \vec{p}_R}{\|\vec{p}_\text{tar} - \vec{p}_R\|} \geq \cos \alpha = \cos \left(\arcsin \left(\frac{r_{\text{obst},i}}{\|\vec{p}_{\text{obst},i} - \vec{p}_R\|}\right)\right)$$  \hspace{1cm} (14)

2. Collision risk, i.e. the robot and obstacle are moving towards each other. This is established by comparing the direction of the position difference vector between the robot and the obstacle with the vector in the direction of the relative velocity.

$$\frac{\vec{p}_{\text{obst},i} - \vec{p}_R}{\|\vec{p}_{\text{obst},i} - \vec{p}_R\|} \cdot \frac{\vec{p}_\text{tar} - \vec{p}_R}{\|\vec{p}_\text{tar} - \vec{p}_R\|} \geq \cos \alpha = \cos \left(\arcsin \left(\frac{r_{\text{obst},i}}{\|\vec{p}_{\text{obst},i} - \vec{p}_R\|}\right)\right)$$  \hspace{1cm} (15)

3. Proximity, i.e. all obstacles within a one meter radius of the robot are considered significant. This value is determined by trial and error.

If an obstacle satisfies one or more of these criteria it is considered relevant and assigned a penalty factor $P_{f,i}$ given in equation (16) which is an exponential function of the relevant distance between the robot and the obstacle. Being one, if obstacle and robot are close and approaching zero if not.

$$P_{f,i} = \min\left(1, \frac{1}{e^{\|\vec{p}_{\text{obst},i} - \vec{p}_R\|-r_{\text{obst},i}}\right)$$  \hspace{1cm} (16)

When an obstacle does not meet either of the criteria anymore its penalty function is not set to zero at once but decays by 0.1 at each subsequent step.
When each obstacle is assigned a penalty factor the global potential field is constructed. It consists of the target function plus the sum of all barrier functions weighed with the appropriate penalty factor. Figure 3.1 shows the constructed field combining the target and barrier functions.

\[ f_{\text{total}} = f_{\text{Target}} + \sum_{i=1}^{NO} P_i \cdot f_{\text{barrier},i} \]  

(17)

In order to navigate around the field of pillars the gradient of the potential field \( f_{\text{total}} \) is calculated \( G_{fp} = \nabla f_{\text{total}} \) which in fact is the artificial force obtained in the potential field method. The direction of the gradient of the field is modified by multiplying it with the inverse of the Hessian function which results in the Newton direction \( \vec{d}_n \). The Hessian matrix contains all second order partial derivatives of the potential field. For detailed information about the Hessian function see appendix A.1.

\[ \vec{d}_n = H_{fp}^{-1} \cdot G_{fp} \]  

(18)

3.1 Solutions for mathematical problems

The first problem is that the target function is a quadratic function which implies that the gradient increases linearly with the distance between the robot and the target position, i.e. far away from the target the robot could be drawn too close towards the obstacles since the target gradient overpowers that of the obstacles. Controversy very close to the target the gradient becomes very flat and the robot might not be able to reach the target if obstacles are also present. In order to prevent this, the target function \( f_{\text{Target}} \) is scaled by dividing it by the distance between the robot and the target raised to a power, as given in equation (19).

\[ W_{t} = \frac{1}{\| \vec{P}_{\text{tar}} - \vec{P}_{\text{r}} \|^W} \]  

(19)

Remark: The barrier functions \( f_{\text{barrier},i} \) increase exponentially and since the size of the environment \( C \) is finite, it is unlikely that the target function \( f_{\text{target}} \) will overpower a barrier function \( f_{\text{barrier},i} \) and cause a collision. Secondly, even with this scaled target function the robot is still not always able to reach the target when it is too close to an obstacle (or edge of the field). Especially when travelling at lower speeds.

A second problem that occurs is that of local minima as seen in Figure 3.2. In certain configurations with one or multiple obstacles local minima exist. Since the Newton direction is in essence a modified gradient descent method it will always steer towards a minimum. So the robot could get trapped in a local minimum which is not the target position. To escape from these local minima Mouton keeps a log of all its previous positions. If the difference between its current position and its position one second ago is less than one meter and the robot has a non-zero penalty, a ‘ghost’ obstacle will be created in its old position, i.e. a virtual obstacle will be created which pushes the robot onwards and out of the local minimum.
3.2 Solutions for physical problems

In the real world the robot has mass and inertia, so acceleration and deceleration are finite and have a value of \( \pm \alpha_{\text{max}} \). This means it takes time for the robot to stop and/or change direction. If the robot travels too close and too fast near an obstacle it cannot avoid a collision any more. To guarantee safe operation the effective maximal velocity of the robot is adjusted downwards if obstacles are present (and significant). The maximal velocity depends on the distance to the nearest obstacle and is given by equation (20).

\[
\vec{V}_{\text{max}} = \vec{V}_{\text{max physical}} \cdot \left( 1 - \left( e^{\text{Max Penalty Factor - Tolerance}} - e^{-\text{Tolerance}} \right)^{\frac{1}{\text{VRF}}} \right) \tag{20}
\]

Where,
\[
\text{Tolerance} = 1 \\
\text{VRF} = 1.51
\]

The calculated direction of motion needs to be translated into an actual movement. Since acceleration is limited also the velocity change in a certain time step is limited which is defined as \( d\vec{V}_{\text{max}} \). This can be converted into two conditions, equations (21) and (22) respectively.

\[
\| d\vec{V} \| \leq dV_{\text{max}} \tag{21}
\]
\[
\| \vec{V} \| + \| d\vec{V} \| \leq V_{\text{max}} \tag{22}
\]

Mouton established three criteria, from which four scenarios can be recognized. For each an appropriate velocity change is computed taking into account the physical limits (21) and (22). A graphical representation of the four scenarios is given in Figure 3.4. The three criteria are

- Accelerating/decelerating; if the smallest angle between the current velocity \( \vec{V}_{\text{cur}} \) and the desired velocity in the Newton-direction \( \vec{V}_{\text{des}} \) is greater than ninety degrees, the velocity change is considered as a deceleration action, if the angle is less than ninety degrees it is considered as an acceleration action.
o Intersect exist; If the Newton direction \( \vec{V}_{des} \) intersects the circle which represents the possible velocity change the robot can obtain the desired velocity direction within one time step. If \( \vec{V}_{des} \) does not intersect this area the robot will not be able to reach the desired with one time step and will chose the velocity change which brings it closest to the desired velocity.

o Velocity limit; the robot cannot/may not exceed its maximum velocity limit

The four scenarios illustrates by Figure 3.4 a to b are:

A. Figure 3.4 a: In this scenario the robot is traveling at a current velocity \( \vec{V}_{cur} \). In order to obtain the desired velocity \( \vec{V}_{des} \) the robot needs to decelerate. The quickest way to establish this is to apply the maximum acceleration in the opposite direction of the current velocity; mathematically this is expressed as equation (23)

\[
d\vec{V} = dV_{\text{max}} \cdot -\frac{\vec{V}_{cur}}{\|\vec{V}_{cur}\|}
\]  

(23)

Mouton lowers the maximum allowable velocity \( V_{\text{max}} \) near obstacles. This could cause the robot to exceed its maximum velocity without the desire to change its direction. Applying the maximum acceleration in the opposite direction of \( \vec{V}_{cur} \) could cause the robot to slow down too much. In this case the following equation (24) is used, which selects the smallest velocity change.

\[
d\vec{V} = \min \left( dV_{\text{max}} \cdot -\frac{\vec{V}_{cur}}{\|\vec{V}_{cur}\|}, V_{\text{max}} \cdot \frac{\vec{d}_{n}}{\|\vec{d}_{n}\|} - \vec{V}_{cur} \right)
\]  

(24)

Where \( \vec{d}_{n} \) is the Newton direction.

B. Figure 3.4 b: The robot is traveling at a velocity \( \vec{V}_{cur} \) but there is no intersect with \( \vec{V}_{des} \) i.e. the robot cannot obtain the desired velocity within one time step. The fastest way to obtain \( \vec{V}_{des} \) is following a line perpendicular to the Newton direction resulting in the following expression

\[
d\vec{V} = \text{sign}(x) \cdot dV_{\text{max}} \cdot \left[ \vec{d}_{n,y} \cdot \vec{d}_{n,y} - \vec{d}_{n,y} \right]
\]  

(25)

Where,

\[
z = z - \text{component of} \begin{bmatrix} \vec{d}_{n,y}, \vec{d}_{n,y}, 0 \end{bmatrix} \times [\vec{u}, \vec{v}, 0]
\]

C. Figure 3.4 c: In this case the desired velocity change is the one that will lead to the largest absolute velocity in the Newton direction. This is done by determining the intersect locations of the Newton Direction vector \( \vec{d}_{n} \) with the maximum velocity change circle. The difference vector between the current velocity \( \vec{V}_{cur} \) and the intersect points is added to \( \vec{V}_{cur} \) from which the largest value is chosen.
D. Figure 3.4 d: If the robot would accelerate past its maximum velocity the, the correct velocity change is the one that will let it travel in the Newton direction, as described by equation (26)

\[ d\dot{V} = V_{\text{max}} \cdot \frac{d_n}{\|d_n\|} - \dot{V} \]  

(26)

Remark: The approach of Mouton will always cause the robot to accelerate to its maximum allowable velocity, if not hindered by obstacles. This is desirable far away from the target; though when the target is near, traveling at maximum velocity will cause the robot to overshoot the target. Secondly, despite velocity and acceleration being limited, inertia and external forces (friction, centripetal) are not taken into account. Simply limiting the maximum acceleration is only justified for a omnidirectional robot.

Another physical problem that Mouton encountered is the finite size of the free space C. Mouton proposed to place an obstacle directly in front of the robot at the border in the direction the robot is travelling. It is placed just outside the border to prevent it from influencing paths parallel to the field boundaries too much.

![Figure 3.5: Placement of a virtual obstacle to prevent the robot from crossing the field boundaries.](image)

![Figure 3.6: Placement of a virtual obstacle to prevent the robot from colliding with another dynamic obstacle.](image)

3.3 Dynamic obstacles

The obstacles the robot encounters during operation do not have to be stationary but could be other moving robots. In order to account for these dynamic obstacles Mouton uses virtual ‘stationary’ obstacles, placed at the position at which the two robots would collide, as seen in Figure 3.6. The real obstacle is rated as insignificant and is ignored. Merely placing a virtual at the intersect point is not sufficient for preventing a collision so it is placed closer towards the robot and shifted towards one side of the real obstacle to effectively steer the robot away from the obstacle.

Remark: This method has several disadvantages; first of all if the robot changes its velocity due to the influence of the virtual obstacle, the collision point with the real obstacle also changes which in turn results in a moving virtual obstacle. Secondly the real obstacle is removed so if the robot accidently steers towards the wrong direction, which could be caused due to a velocity change of the obstacle, a collision cannot be avoided any more. Third of all in the chapter ‘Future work’, Mouton states that this method is not sufficient in preventing a head on collision at maximum velocity \( \dot{V}_{\text{max}} \) which is caused by the movement of the virtual obstacle.
4. Extensions upon the potential field method

Simply using the potential field method as explained in Chapter two is not sufficient to always find a collision free path towards the target. Therefore, some additional methods are developed based to improve the functioning of the potential field method.

In the real world the target and obstacles are likely to be non-stationary. Therefore it is desirable to take the relative velocity between robot and target and robot and obstacles into account in the computation of the most promising direction of movement $\vec{F}$. GE and Cui (2002) proposed a modified potential field method which makes full use of the position and velocity information of the robot, target and obstacles. The attractive potential function is given by equation (27).

$$U_{att}(\vec{p}_R, \vec{v}_R) = \alpha_p \rho_T^m + \alpha_v v_{RT}^n$$  \hspace{1cm} (27)

Where $\alpha_p$ and $\alpha_v$ are positive scaling factors, $m$ and $n$ are positive scalar parameters, $\rho_T$ is the Euclidean distance between the robot and the target and $v_{RT}$ is the relative velocity between robot and target.

$$\rho_T = \|\vec{p}_{RT}\| = \|\vec{p}_{tar} - \vec{p}_R\| = \vec{p}_{tar} - \vec{p}_R \cdot \vec{n}_{RT}$$  \hspace{1cm} (28)

$$v_{RT} = \|\vec{v}_{RT}\| = \|\vec{v}_{tar} - \vec{v}_R\| = \vec{v}_{tar} - \vec{v}_R \cdot \vec{n}_{VRT}$$  \hspace{1cm} (29)

Where: $\vec{p}_{tar}$, $\vec{p}_R$, $\vec{v}_{tar}$ and $\vec{v}_R$ are the target and robot position and velocity, respectively. $\vec{n}_{RT} = \frac{\vec{p}_{RT}}{\|\vec{p}_{RT}\|}$ is a unit vector pointing from the robot to the target and $\vec{n}_{VRT} = \frac{\vec{v}_{RT}}{\|\vec{v}_{RT}\|}$ is a unit vector pointing in the direction of the relative velocity between robot and target.

The influence of the parameters $m$, $n$, $\alpha_p$ and $\alpha_v$ on the system is discussed in detail in appendix A.2. Note that if $\alpha_v = 0$ and $m = 2$ (27) reduces to the conventional (stationary) form (30). Moreover, when $\alpha_p = 1$, $\alpha_v = 0$ and $m = 2$, it reduces to the target function $f_{target}$ used by Mouton.

$$U_{att}(\vec{p}_R) = \alpha_p \rho_T^2$$  \hspace{1cm} (30)

From (27) the corresponding virtual attractive force can be derived, which is defined as the negative gradient of the attractive potential function (31). Since (27) is differentiable for every $x$ and $y$, (27) is also differentiable for every combination of $x$ and $y$, i.e. every configuration $\vec{p}_R$. Similar holds for the velocity component $\vec{v}_R$. So $\vec{F}_{att}$ can be expressed in terms of position $\vec{p}_R$ and velocity $\vec{v}_R$.

$$\vec{F}_{att}(\vec{p}_R, \vec{v}_R) = -\nabla U_{att}(\vec{p}_R, \vec{v}_R) = -\nabla U_{att}(\vec{p}_R) - \nabla U_{att}(\vec{v}_R)$$

$$= -\left(\frac{\partial}{\partial \vec{p}_R} + \frac{\partial}{\partial \vec{v}_R}\right) U_{att}(\vec{p}_R, \vec{v}_R) = -\frac{\partial U_{att}(\vec{p}_R, \vec{v}_R)}{\partial \vec{p}_R} - \frac{\partial U_{att}(\vec{p}_R, \vec{v}_R)}{\partial \vec{v}_R}$$  \hspace{1cm} (31)

When choosing the parameters $m$ and $n$ in equation (27) it must be kept in mind that when $\vec{p}_R = \vec{p}_{tar}$ (27) is non-differentiable for $m = 1$ with respect to position, and when $\vec{v}_R = \vec{v}_{tar}$ it is non-differentiable for $n = 1$. This causes singularities in the algorithm in implementation. Secondly the parameter $n$ has to be larger than one if a soft landing approach is required, i.e. $\vec{v}_R = \vec{v}_{tar}$ when the target position is reached. If a hard landing approach is allowed, $\vec{v}_R \neq \vec{v}_{tar}$ at target position, there is no such constrained as long as $m$ and $n > 0$.

When $\vec{p}_R \neq \vec{p}_{tar}$ and $\vec{v}_R \neq \vec{v}_{tar}$ the total attractive force can be derived

$$\vec{F}_{att}(\vec{p}_R, \vec{v}_R) = \vec{F}_{att1}(\vec{p}_R) + \vec{F}_{att2}(\vec{v}_R)$$  \hspace{1cm} (32)
Where,

\[ \vec{F}_{\text{attr}}(\vec{R}_R) = m\alpha_p \rho_{RT}^{n-1} \cdot \vec{n}_{RT} \]  \hspace{1cm} (33)

\[ \vec{F}_{\text{attr}}(\vec{v}_R) = n\alpha_v \rho_{VRT}^{n-1} \cdot \vec{n}_{VRT} \]  \hspace{1cm} (34)

In Figure 4.1 a graphical representation is given of the virtual attractive force computation in 2-D space.

In a similar fashion a repulsive potential function can be defined which also makes full use of position and velocity information. The relative velocity between the robot and an obstacle in the direction from the robot to the obstacle is given by equation (35).

\[ v_{ROI} = (\vec{v}_{\text{obst},i} - \vec{v}_R) \cdot \vec{n}_{ROI} \]  \hspace{1cm} (35)

Where \( \vec{n}_{ROI} \) is a unit vector pointing from the robot to the target. If \( v_{ROI} \geq 0 \) the robot is moving away from obstacle \( i \) so no avoiding action needs to be undertaken, contrary if \( v_{ROI} < 0 \) the robot is moving towards obstacle \( i \) and an avoiding action should be undertaken. As said before the physical acceleration of the robot is limited which implies that the robot travels a certain distance \( \rho_m \), with respect to obstacle \( i \), before the relative velocity \( v_{ROI} \) reduces to zero, which is required to avoid the obstacle.

\[ \rho_m = \frac{-v_{ROI}^2}{2a_{\text{max}}} \]  \hspace{1cm} (36)

If now the shortest distance between the robot and the obstacle is defined as \( \rho_R \), the repulsive potential can be written as:

\[ U_{\text{rep},i}(\vec{p}_R, \vec{v}_R) = \begin{cases} 
0 & \text{if } \rho_{s,i} - \rho_R - \rho_m \geq \rho_{bi} \text{ or } v_{ROI} \geq 0 \\
\eta \left( \frac{1}{\rho_{s,i} - \rho_R - \rho_m} - \frac{1}{\rho_{bi}} \right) & \text{if } 0 < \rho_{s,i} - \rho_R - \rho_m < \rho_{bi} \\
\text{Not defined} & \text{if } v_{ROI} < 0 \text{ and } \rho_{s,i} - \rho_R < \rho_m 
\end{cases} \]  \hspace{1cm} (37)

Where \( \eta \) is a positive scaling constant and \( \rho_{bi} \) defines the influence range of obstacle \( i \). When \( \rho_{s,i} - \rho_R < \rho_m \) the repulsive function is not defined since there is no possible solution to avoid a collision. While the robot is far away from the obstacle, i.e. \( \rho_{s,i} - \rho_R - \rho_m \geq \rho_{bi} \) or \( v_{ROI} \geq 0 \), the robot is not influenced by the obstacle and no avoiding action is implemented. If the robot is within the influence range of the obstacle and \( \rho_{s,i} - \rho_R - \rho_m \) approaches zero, the repulsive potential approaches infinity. A larger relative velocity \( v_{ROI} \) results in a larger value of \( \rho_m \) the sooner the criteria \( \rho_{s,i} - \rho_R - \rho_m(v_{ROI}) < \rho_{bi} \) is met the sooner the robot is influenced by the obstacle.
Remark: When the repulsive function is not defined, i.e. a collision is unavoidable, a maximum acceleration should be applied in the direction opposite of the relative velocity between robot and obstacle to reduce the collision speed. The physical size of the robot $\rho_R$ can be adapted to accommodate for measurement uncertainty in the velocity and position information, also the physical size of the obstacles can be added.

When the repulsive potential function is known the repulsive force can be calculated by taking the derivative of the repulsive potential function which results in the two repulsive forces $\vec{F}_{\text{rep},1,i}$ and $\vec{F}_{\text{rep},2,i}$.

$$
\vec{F}_{\text{rep},1,i} = -\nabla U_{\text{rep},i}(\vec{p}_R, \vec{v}_R) \\
\vec{F}_{\text{rep},2,i} = -\nabla^2 U_{\text{rep},i}(\vec{p}_R, \vec{v}_R) - \nabla \vec{v}_R U_{\text{rep},i}(\vec{p}_R, \vec{v}_R) = \vec{F}_{\text{rep},1,i} + \vec{F}_{\text{rep},2,i}
$$

(38)

$$
\vec{F}_{\text{rep},i}(\vec{p}_R, \vec{v}_R) = \begin{cases} 
0 & \text{if } \rho_{s,i} - \rho_R - \rho_m \geq \rho_{oi} \text{ or } v_{ROI} \geq 0 \\
\vec{F}_{\text{rep},1,i} + \vec{F}_{\text{rep},2,i} & \text{if } 0 < \rho_s - \rho_i - \rho_m < \rho_{oi}
\text{and } v_{ROI} < 0 \\
\text{Not defined} & \text{if } v_{ROI} < 0 \text{ and } \rho_{s,i} - \rho_R < \rho_m
\end{cases}
$$

(39)

Where,

$$
\vec{F}_{\text{rep},1,i}(\vec{p}_R, \vec{v}_R) = -\frac{\eta}{(\rho_{s,i}-\rho_R-\rho_m)^2} \left( 1 + \frac{v_{ROI}}{d_{\text{max}}} \right) \vec{n}_{ROI,i}
$$

(40)

$$
\vec{F}_{\text{rep},2,i}(\vec{p}_R, \vec{v}_R) = \frac{\eta v_{ROI} \rho_{ROI}}{\rho_s \rho_{\text{max}}(\rho_s - \rho_m)^2} \vec{n}_{ROI,i}
$$

(41)

When the attractive force $\vec{F}_{\text{att}}$ and the repulsive forces are known the total force can be obtained. For multiple obstacles the total force $\vec{F}$ is given by equation (42), where $n_{\text{obs}}$ is the number of obstacles.

$$
\vec{F} = \vec{F}_{\text{att}} + \sum_{i=1}^{n_{\text{obs}}} \vec{F}_{\text{rep},i}
$$

(42)

### 4.1 Oscillation problems

A disadvantage of the potential field method is that it tends to oscillate in the presence of obstacles or in narrow passages (Koren and Borenstein (1991)). This problem can cause slow progress or system instability in implementation. Ren and McIsaac (2006) derived a modified Newton’s Method using the Hessian function $H$, similar to the one Mouton used, to solve this problem. Their method reduces oscillations by finding an appropriate direction for the quadratic approximation of the potential function instead of the linear approximation generally used, i.e. the components of the Hessian function are a measurement of the curvature of the field. The Hessian function modifies the direction of the calculated artificial force; as a result the robot steers away from highly curved places in the field. For example a high curvature in $x$ direction will result in a small value of the $H^{-1}(1,1)$ component of the inverse Hessian function and shift the artificial force vector towards the $y$ component.

$$
\vec{F} = -H^*(\vec{p}_R)^{-1} \frac{dU(\vec{p}_R)}{d\vec{p}_R}
$$

(43)

Where,

$$
H^* = H + \mu I
$$

, Positive definite scaled Hessian

$$
H = \begin{bmatrix}
\frac{\partial^2 U(\vec{p}_R)}{\partial x^2} & \frac{\partial^2 U(\vec{p}_R)}{\partial x \partial y} \\
\frac{\partial^2 U(\vec{p}_R)}{\partial x \partial y} & \frac{\partial^2 U(\vec{p}_R)}{\partial y^2}
\end{bmatrix} \in \mathbb{R}^2
$$

, Hessian function

It must be noted that introducing the inverse of $H$ is unattractive for most optimization problems since it scales with the dimensionality raised to the power of two. This makes the computational cost for high dimensional
problems rather expensive. However for a low dimensional problem, such as addressed in this paper, the additional computational costs are negligible. Secondly the Hessian needs to be positive definite for its inverse to exist. This can be guaranteed by using the solution of Papalambros and Wilde (2000), appendix A.1. The research of Ren and McIsaac proved three advantages of using the Hessian function in the path planning algorithm for a mobile robot moving in a plane; it reduces task completion time; generates smoother trajectories with little oscillations and allows for a bigger step size than without the Hessian.

Remark: In this notation the Hessian function contains the derivatives to $x$ and $y$. In this paper the potential function is a function of position $\vec{p}_r$ and velocity $\vec{v}_g$. In Appendix A.1 a formulation is given which can be applied on the potential field method of GE and Cui (2002) which used the velocity information in the construction of the field. The second order derivatives are derived in appendix A.4.

4.2 Local and global minima problems

A simple solution for escaping local minima was established by Vadakkepat and Tan (2000). Local minima occur at null-potential areas, so equation (44) must hold.

$$\|\vec{F}_{total}\| = \|\vec{F}_{att} + \sum_{i=1}^{N_o} \vec{F}_{rep,i}\| = 0$$

(44)

A local minimum can be identified when conditions (45) and (46) hold. In (45) and (46), $b$ and $c$ are arbitrary chosen constants.

$$\frac{\|\vec{F}_{total}\|}{\sum_{i=1}^{N_o} \vec{F}_{rep,i}} < b$$

(45)

$$\cos(\alpha_{att} - \sum_{i=1}^{N_o} \alpha_{rep,i}) < -\cos(c)$$

(46)

Assuming $b$ and $c$ have values in the order of 0.1 and $\frac{5}{6}\pi$ respectively, Figure 4.2 shows two situations. In situation A the angles $\alpha_{att}$ and $\alpha_{rep}$ are approximately the same (except for the sign($\pm$)), so condition (46) could be satisfied. However condition (45) will not be satisfied since $\|\vec{F}_{total}\| \approx \|\vec{F}_{rep}\|$, so no local minimum is identified. In case B condition (45) and (46) are both satisfied so a local minimum is present.

![Figure 4.2: Two cases: one where no local minimum can be identified (A) and one where a local minimum can be identified (B).](image)

When a local minimum is near, an additional escape force $\vec{F}_e$ is applied in the direction perpendicular to $\vec{F}_{rep}$. As visualized in Figure 4.3 the escape force effectively steers the robot away from the local minimum and obstacle. The escape force is defined as

$$\vec{F}_e = \frac{1}{p_{i,l}} \vec{n}_{rep,l}$$

(47)
To assure the robot can always reach the target even when obstacles are near and when the robot is travelling at lower velocity, GE and Cui (2000) modified the potential functions. Their modification ensures that the target position always is the global minima of the field. In order to do so, the distance between the robot and the target $\rho_T$ is introduced into the repulsive function. As given in equation (48) the repulsive function is multiplied with the distance from the robot to the target $\rho_T$ raised to a power $k$.

$$U_{rep,i}(\vec{p}_R) = \begin{cases} 0 & \text{if } \rho_{s,i} - \rho_R \geq \rho_{0i} \\ \eta \left(\frac{1}{\rho_{s,i} - \rho_R} - \frac{1}{\rho_{0i}}\right) \rho_R^k & \text{if } 0 < \rho_{s,i} - \rho_R < \rho_{0i} \\ \text{Not defined} & \text{if } \rho_{s,i} < \rho_R \end{cases} \quad (48)$$

From which the repulsive force can be derived

$$\vec{F}_{rep,i}(\vec{p}_R) = -\nabla U_{rep}(\vec{p}_R) = \vec{F}_{rep1i}\vec{n}_{OR,i} + \vec{F}_{rep2i}\vec{n}_{RT} \quad (49)$$

Where,

$$\vec{F}_{rep1i} = \eta \left(\frac{1}{\rho_{s,i} - \rho_R} - \frac{1}{\rho_{0i}}\right) \rho_R^k \quad (50)$$

$$\vec{F}_{rep2i} = \frac{k}{2} \eta \left(\frac{1}{\rho_{s,i} - \rho_R} - \frac{1}{\rho_{0i}}\right)^2 \rho_R^{k-1} \quad (51)$$

In which $\vec{n}_{OR,i} = -\nabla \rho_s$ and $\vec{n}_{RT} = -\nabla \rho_T$ are two unit vectors pointing from the obstacle to the robot and from the robot to the target, respectively. $k$ is a positive constant which determines the mathematical properties of the repulsive function.

- $0 < k < 1$, As the robot approaches the target the first component $\vec{F}_{rep1i}$ approaches zero while the second component $\vec{F}_{rep2i}$ approaches infinity. As a result the robot is strongly pulled towards the target and ignores the obstacle which might results in over shooting the target position and colliding with the present obstacle.
- $k = 1$, In this case when the robot approaches the goal the first component of the repulsive force $\vec{F}_{rep1i}$ approaches zero and the second component $\vec{F}_{rep2i}$ approaches a constant, which again might result in overshooting.
- $k > 1$, the repulsive potential function is differentiable at the target position and as the robot approaches the target position, the total force continuously approaches zero.

Accordingly to Ge and Cui (2000) there is a relationship between $\alpha_p$ and $\eta$ which can eliminate all local minima and guarantee that the target position is the only minimum. However since the real world is highly dynamic and local minima will not exist at the same position for a long time, it is unlikely that the robot will be trapped in one of them. So optimising $\alpha_p$ and $\eta$ at each step is a rather expensive method. Moreover, if the robot does get trapped, it is simple to escape by using the solution of Vadamkepat and Tan (2000).
4.3 Modeling of Field boundaries and non-circular obstacles
In literature (Gilbert and Daniel (1988), Koren and Borenstein (1991)) non-circular objects are often modeled as arrays of points. In this particular case the space boundaries can be modeled in a similar way, Figure 4.4 shows a point configuration and the resulting potential field. To compute this potential field equation (37) is used, where \( \rho_\text{e} = \rho_\text{m} = 0 \). The shortest distance to the obstacle \( \rho_\text{e} \) is calculated at each position of the field. This is done by calculating the distance from the robot position to each point on the surface of the object, from which the smallest value is chosen. The closest point is considered as an obstacle which cannot be crossed.

Figure 4.4: Point configuration and resulting potential field.

4.4 Dynamical model for a holonomic robot with physical limitations
During navigation it is desirable that the robot tracks the gradient of the field accurately. However in the real world acceleration and velocity are finite. Thus, in the algorithm velocity and acceleration should also be finite. In order to do so Utkin and Guldner (2009) developed a sliding mode control system. The goal is to assure that the current velocity \( \vec{V}_\text{cur} \) equals the desired velocity \( \vec{V}_\text{des} \), i.e. minimizing the error \( d\vec{V} \) given in equation (52).

\[
d\vec{V} = \vec{V}_\text{cur} - \vec{V}_\text{des} = \vec{V}_\text{cur} - V_\text{des} \left( \frac{\hat{\rho}}{\max(\|\hat{\rho}\|, \varepsilon)} \right)
\]

Where \( \hat{\rho} / \| \hat{\rho} \| \) is the field gradient scaled to unit length, \( \varepsilon \) is an arbitrarily chosen small value which is introduced to prevent singularities at minima positions. An appropriate choice for \( V_\text{des} \) is given by (53), where \( V_\text{max} \) is the maximum velocity of the robot and \( (2a_\text{max} \rho_\text{T})^{\frac{1}{2}} \) is the ‘brake time’ of the robot needed to stop at the target.

\[
V_\text{des} = \min \left( V_\text{max}, \left( 2a_\text{max} \rho_\text{T} \right)^{\frac{1}{2}} \right)
\]

Where, \( \rho_\text{T} = || \vec{p}_\text{cur} - \vec{p}_\text{T} \| \) is the Euclidean distance between the robot and the target. Beginning at \( t = 0 \), this choice of \( V_\text{des} = V_\text{max} \) ensures the robot will accelerate towards \( V_\text{max} \) with maximum acceleration \( a_\text{max} \). Figure 4.5 illustrates the error \( d\vec{V} \) occurring. In order to converge towards the desired velocity \( \vec{V}_\text{des} \) an acceleration is applied in the direction of \( d\vec{V} \).

Figure 4.5: Error between \( \vec{V}_\text{des} \) and \( \vec{V}_\text{cur} \).
When the robot has reached its maximum velocity $V_{max}$, the choice $V_{des} = V_{max}$ ensures that the robot will never exceed that velocity. If the robot approaches the target, the selection $V_{des} = \sqrt{2a_{max} \rho}$ lowers the desired velocity as a function of the distance between the robot and the target. Enabling the robot to stop at the target position and prevent overshoot.

In order to determine the accelerations applied during operation, the dynamic model of the robot should be considered. Equation (54) gives a general dynamic model for holonomic mechanical systems.

$$M \ddot{a}_{app} + \dot{N}(\ddot{p}_R, \ddot{v}_R) = \ddot{r}$$  \hspace{1cm} (54)

Where; $M$ is the positive definite inertia matrix; $\dot{N}(\ddot{p}_R, \ddot{v}_R)$ represent the body/external forces like gravitation, centripetal and fiction and $\ddot{r}$ is the applied control force. From (54) the required acceleration $\ddot{a}_{app}$ can be obtained by rewriting it into the following form:

$$\ddot{a}_{app} = M^{-1} \left( \ddot{r} - \dot{N}(\ddot{p}_R, \ddot{v}_R) \right)$$  \hspace{1cm} (55)

Where $\ddot{r}$ is defined as:

$$\ddot{r} = \tau_0 \left( \frac{dV}{max(\|dV\| \epsilon)} \right)$$  \hspace{1cm} (56)

Where $\tau_0$ is the maximum available control force; $\frac{dV}{\|dV\|}$ is the unit vector pointing in the direction of $dV$ and $\epsilon$ an arbitrarily chosen small value to prevent singularities in case $dV = 0$. The required acceleration can simply be calculated. This definition of $\ddot{r}$ ensures the control force is applied in the direction of $dV$ which ensures fast convergence towards the desired velocity.

Note that when body forces are neglected, $N(\ddot{p}_R, \ddot{v}_R) = 0$ and a maximal error occurs, $\frac{dV}{\|dV\|} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\ddot{a}_{app}$ becomes:

$$\ddot{a}_{app} = M^{-1} \cdot \ddot{r}_0 = \ddot{a}_{max}$$  \hspace{1cm} (57)

Equation (57) implies that the applied acceleration $\ddot{a}_{app}$ will never exceed $\ddot{a}_{max}$. The calculated acceleration $\ddot{a}_{app}$ can be simply converted into a velocity change by multiplying with the time step taken; $\dot{V}_{app} = \ddot{a}_{app} \cdot dt$.

Remark: The calculated total force $\ddot{F}_{total}$ could also be directly converted into an acceleration limited with a maximum value. However this acceleration would be aligned parallel to the desired Newton direction and will not result in accurate gradient tracking. In further implementation it should be noted that equation (27) and condition (53) both cause the robot to slow down near the target and might influence one another.
5. A comparison between methods

The basic concept behind Mouton and the potential field method is the same. The robot is attracted by the target and repulsed by obstacles. However there are some differences in the definition of the attractive and repulsive potentials. Equation (58) is the general definition of the attractive potential $U_{att}$. Equation (59) denotes the target function $f_{target}$ (attractive potential) used by Mouton.

$$U_{att}(\vec{p}_R) = \alpha_p \rho_{T}^n$$  \hspace{1cm} (58)

$$f_{target} = ||\vec{p}_{tar} - \vec{p}_R||^2 = \rho_{T}^2$$  \hspace{1cm} (59)

The target function $f_{target}$ of Mouton has the shape of a quadratic well ($m = 2$), which implies that the attractive force will linearly decline until zero when the robot approaches the target. Moreover, it ensures that $f_{target}$ is differentiable at the target position which prevents singularities. So the choice of using $m = 2$ is good. Now look at the definition of the repulsive potential $U_{rep,i}$ of the general potential field method given by equation (60) and the barrier function $f_{barrier,i}$ used by Mouton (61).

$$U_{rep,i}(\vec{p}_R) = \left\{ \begin{array}{ll} 0 & \text{if } \rho_{s,i} - \rho_R > \rho_{0i} \\ \eta \left( \frac{1}{\rho_{s,i} - \rho_R} - \frac{1}{\rho_{0i}} \right) & \text{if } \rho_{s,i} - \rho_R \leq \rho_{0i} \end{array} \right.$$  \hspace{1cm} (60)

$$f_{barrier,i} = \frac{1}{1 - \left( \frac{1}{\rho_{barrier,i} - \rho_R} \right)}, \ i = 1, \ldots, N_0$$  \hspace{1cm} (61)

The repulsive potential $U_{rep,i}$ is zero for all obstacles for which the criteria $\rho_{s,i} - \rho_R > \rho_{0i}$ holds. Only the obstacle for which the criteria $\rho_{s,i} - \rho_R \leq \rho_{0i}$ holds influence the robot. Mouton computes the repulsive potential field for each obstacle and then takes the sum of all repulsive functions weighted with a penalty function $P_{i,t}$, which in fact is double work. So the criteria used in equation (60) are preferred and will be implemented.

In chapter four the introduction of the velocity information into the potential field $U$ is discussed. Compared to Mouton this has several advantages:

- First of all Mouton is an un-damped system which implies that the robot will oscillate around the target position at its natural frequency $\omega_n$ as seen in Figure 5.1. This can be understood using the work of GE and Cui (2002). They derived the damping ratio $\xi$ and the natural frequency $\omega_n$ for a holonomic omnidirectional robot given by equations (62) and (63).

$$\xi = \frac{\alpha_p'}{2\alpha_p}$$  \hspace{1cm} (62)

$$\omega_n = \sqrt{2\alpha_p'}$$  \hspace{1cm} (63)

Where: $\alpha_p' = \frac{a_x}{m_p}$, $\alpha_p' = \frac{a_y}{m_p}$ with $m_p$ the mass of the robot. GE and Cui (2002) investigated the influence of the parameters $m$, $n$, $\alpha_p$ and $\alpha_v$ (appendix A.2). For $0 < \xi < 1$ the system is under-damped; for $\xi = 1$ the system is critically damped and for $\xi > 1$ the system is over-damped. This also allows for a controlled approach of the target. Since Mouton does not use the velocity information, i.e. $\alpha_v = 0$, it will always be an un-damped system and will not slow down when approaching the target.
Using the velocity information in the repulsive potential function ensures that robot always steers away from possible collision points. This makes the placement of virtual obstacles at the collision point redundant, which in turn eliminates the risk of ending up in a real obstacle while avoiding a virtual obstacle. Additionally, the navigation becomes less expensive since the placement of a virtual obstacle and then shifting it towards one side of the collision point is rather cumbersome.

Mouton is not able to efficiently trace a dynamic target. In Mouton the robot is attracted towards the current position of the target and not to the position it will be in after a few time steps. GE and Cui’s method follows the shortest route to intercept the target. The velocity information is also used in the computation of the repulsive potential field $U_{rep,i}$. Therefore there is no need to lower the maximum allowable velocity of the robot near obstacles. This can be understood by looking at the criteria $\rho_{s,i} - \rho_R - \rho_m < \rho_{ol}$. A high relative velocity $v_{ROI}$ will result in a large breaking distance $\rho_m$. So the criteria $\rho_{s,i} - \rho_R - \rho_m < \rho_{ol}$ will hold for larger $\rho_{s,i}$, i.e. the higher $v_{ROI}$ the sooner the robot will be influenced by obstacle $i$.

Since the usage of the velocity information in the potential field $U$ enhances the performance of the algorithm with respect to obstacle avoidance and target tracing it will be implemented into the new algorithm.

### 5.1 Local and Global minima

As discussed in section 3.1 the target position is not always the global minimum of the field which prevents the robot from reaching the target. Figure 5.2 shows the field lines of the potential field and the distance between robot and obstacle and robot and target. First the robot overshoots the target position; then it collides with the obstacle; overshoots the target position again and gets trapped in the global minimum which prevents the robot from reaching the target. Mouton tried to solve this problem by scaling the target function (eq. (19)) with the distance between robot and target raised to a power, however this is not sufficient since it still occurs. GE and Cui (2000) did a similar thing by scaling the repulsive potential function (eq (46)) which ensured that the target position is always the global minimum of the entire field. The difference between them is not only the function which is scaled; L Mouton treats the weigh factor $W_t$ as a constant when the derivatives are taken of the target function, which is mathematically incorrect since it is a function of position. GE and Cui do not treat this factor as a constant when the derivative of the repulsive potential function is taken but also differentiate it. As a result the robot is always able to reach the target even when it is close to an obstacle.
To solve the problem of local minima, Mouton uses a ghost obstacle to push the robot onwards when it is trapped in one. This solution allows the robot to escape the local minimum, however progress is often slow. The solution of Vadakkepat and Tan (2000) uses an escape force $F_e$ to prevent a robot from entering a local minima, this solution has a couple of advantages.

- No extra local minima are created; as seen in Figure 3.3 the ghost obstacle amplifies the local minimum by enclosing the robot between three obstacles. Moreover, it pushes the robot towards the real obstacles. This slows down the robot significantly.
- Faster progress is possible; the escape force is applied as soon as a robot gets within a predefined distance from a local minimum. A ghost obstacle is only created when the robot has moved less than one meter within the last second, so it has to wait for one second to escape a minimum.
- No record of the old positions has to be hold, so less memory is required.

Scaling the repulsive potential $U_{rep}$ with the distance from the robot towards the target ensures the robot will always be able to reach the target when it is near to an obstacle. The usage of the escape force $F_e$ enables the robot to quickly escape local minima. Since both methods improve functioning of the algorithm they will used for implementation.

### 5.2 Desired direction of motion computation

In literature, generally the artificial force $\hat{F}$, which is defined as the negative gradient of the potential field $U$, is considered as the most promising direction of motion. Mouton modifies this direction by multiplying it with the inverse of the Hessian function which results in the Newton Direction. Ren and McIsaac (2006) proved the Hessian function reduces oscillation problems during navigation, provides smoother trajectories and allows for a greater step size in the general potential field method. Since this paper considers a low dimensional problem extra computing costs are negligible, therefore the application of the Hessian is desirable.

### 5.3 Physical limitations and field boundaries

The basic concept which determines the velocity change of the robot is quite the same in Mouton and Utkin and Guldner (2009). Both are based on reducing the error between the current and desired velocity. However the implementation is quite different. Utkin and Guldner (2009) consider the dynamical model of the robot, Mouton does not. Mouton simply limits acceleration by limiting the maximum velocity change that can be accomplished within one time step. Considering the dynamical model of the robot allows for the integration of the inertia matrix of the robot and external/body forces in the calculation of the desired acceleration. This provides a more realistic representation of the robots behavior. (The Mouton method is only valid for an omnidirectional robot)
A second problem that occurs in Mouton was explained in section 3.2: the robot, if not hindered by obstacles, always accelerates to its maximum velocity which causes the robot to overshoot the target. The sliding mode control method does not have this problem since it lowers the desired velocity near the target. Another advantage of the sliding mode control is that it efficiently tracks the gradient directly from the error information without exceeding the maximum velocity or acceleration. No intersections need to be calculated and no criteria need to be checked, it always converges towards the desired direction and velocity.

Despite both methods control the velocity vector, Mouton calculates a velocity change while Utkin and Guldner calculate an acceleration which needs to be applied in order to reduce the velocity error. In the computer model this does not make a difference, since the applied acceleration is converted into a velocity change for visualization. However when a real robot needs to be controlled it is easier to obtain the required acceleration since real actuators apply a force which can be easily related to the required acceleration by equation (55). Moreover, a velocity change of the robot is measured by measuring the acceleration. So it can be checked directly whether or not the applied force generates the required acceleration.

The field boundaries are accounted for in Mouton by using a virtual obstacle just behind the boundary which prevents the robot from crossing it. Another method to model the field boundary is as an array of points as explained in section 4.3. A great benefit of Mouton’s method is that the obstacle is placed directly in front of the robot in the direction of motion. This allows the robot to travel parallel to a boundary without being influenced by it, since no obstacle is placed there as it will never cross the parallel boundary. By introducing the relative velocity into the potential field function of the point array method a similar result could be obtained, though this increases computational costs, since not only the distance needs to be calculated but also the relative velocity. Which points out another advantage of Mouton, there is no need to calculate the distance to all surrounding points on the field boundary, which reduces computational costs. A shortcoming of Mouton is that it cannot model non-circular objects because it assumes that all obstacles are circular. By adding the point-array method this problem could be solved.
6. Implementation concept

To improve the functioning of the robot, with respect to obstacle avoidance and target tracing, the sections 4.2 and 4.4 are implemented into the code of Mouton, which implies that the velocity information is used within the potential field computation and that the repulsive potential function is scaled with the distance from the robot to the target $\rho_T$ raised to a power; the resulting artificial forces are modified with the inverse Hessian function; an escape force is used to escape local minima and the dynamic model of the robot is introduced to replace the solution of Mouton.

6.1 The desired direction of motion

As said above two changes are made in the definition of the potential field $U$. The velocity information is used in the attractive potential $U_{\text{att}}$ as well as in the repulsive potential $U_{\text{rep}}$. Moreover, the repulsive potential $U_{\text{rep}}$ is scaled with the distance from the robot to the target $\rho_T$ raised by a power $k$. Combining these two solutions yields the following formulation of the potential field $U$:

$$U(\vec{p}_R, \vec{v}_R) = U_{\text{att}}(\vec{p}_R, \vec{v}_R) + U_{\text{rep}}(\vec{p}_R, \vec{v}_R)$$  \hfill (64)

Where the attractive potential $U_{\text{att}}$ is given by

$$U_{\text{att}}(\vec{p}_R, \vec{v}_R) = \alpha_p \rho_T^m + \alpha_v v_{RT}^n$$  \hfill (65)

which is the same formula as equation (27) in Chapter four. The repulsive potential $U_{\text{rep}}$ is given by

$$U(\vec{p}_R, \vec{v}_R) = \sum_{i=1}^{n_{\text{obs}}} U_{\text{rep},i}$$  \hfill (66)

Where

$$U_{\text{rep},i}(\vec{p}_R, \vec{v}_R) = \begin{cases} 0 & \text{if } \rho_{s,i} - \rho_{R,i} - \rho_m \geq \rho_{0,i} \text{ or } \rho_{R,i} \geq 0 \\ \eta \left( \frac{1}{\rho_{s,i} - \rho_{R,i} - \rho_m} - \frac{1}{\rho_{0,i}} \right) \rho_T^k & \text{if } 0 < \rho_{s,i} - \rho_{R,i} - \rho_m < \rho_{0,i} \\ \text{Not defined} & \text{if } \rho_{R,i} < 0 \text{ and } \rho_{s,i} - \rho_R < \rho_m \end{cases}$$  \hfill (67)

In the middle of equation (67) the introduced scaling factor $\rho_T^k$ is seen. The desired direction of motion (artificial force) is determined by taking the negative gradient of the complete potential field $U$ which is a function of the position of the robot $\vec{p}_R$ and the velocity of the robot $\vec{v}_R$. So the artificial force can be written as:

$$\vec{F} = -\nabla U(\vec{p}_R, \vec{v}_R) = -\frac{\partial U(\vec{p}_R, \vec{v}_R)}{\partial \vec{p}_R} - \frac{\partial U(\vec{p}_R, \vec{v}_R)}{\partial \vec{v}_R} = \vec{F}_p + \vec{F}_v$$  \hfill (68)

Two artificial forces are defined; $\vec{F}_p$, which represents the artificial force with respect to position and $\vec{F}_v$, which represents the artificial force with respect to velocity. These artificial forces can be further separated in their attractive and repulsive components as given in (69) and (70).

$$\vec{F}_p = -\frac{\partial U(\vec{p}_R, \vec{v}_R)}{\partial \vec{p}_R} = -\frac{\partial U_{\text{att}}(\vec{p}_R, \vec{v}_R)}{\partial \vec{p}_R} - \frac{\partial U_{\text{rep}}(\vec{p}_R, \vec{v}_R)}{\partial \vec{p}_R} = \vec{F}_{\text{att},p} + \vec{F}_{\text{rep},p}$$  \hfill (69)

$$\vec{F}_v = -\frac{\partial U(\vec{p}_R, \vec{v}_R)}{\partial \vec{v}_R} = -\frac{\partial U_{\text{att}}(\vec{p}_R, \vec{v}_R)}{\partial \vec{v}_R} - \frac{\partial U_{\text{rep}}(\vec{p}_R, \vec{v}_R)}{\partial \vec{v}_R} = \vec{F}_{\text{att},v} + \vec{F}_{\text{rep},v}$$  \hfill (70)

Where the position and velocity components of the attractive force $\vec{F}_{\text{att},p}$ and $\vec{F}_{\text{att},v}$ respectively, are denoted by:
The coordinate and velocity components of the repulsive force \( \vec{F}_{\text{rep}} \) are:

\[
\vec{F}_{\text{rep}} = \sum_{i=1}^{n_{\text{obs}}} \left( \frac{-\eta \rho_i^k}{(\rho_i - \rho_R - \rho_m)^2} \vec{n}_{RO} + \frac{\eta v_{RO} v_{ROL} \rho_i^k}{\rho_i a_{\text{max}} (\rho_i - \rho_R - \rho_m)} \vec{n}_{RRO} + k \eta \left( \frac{1}{\rho_i - \rho_R - \rho_m} - \frac{1}{\rho_0} \right) \rho_i^k \vec{n}_{RT} \right)
\]

(73)

\[
\vec{F}_{\text{rep}, v} = \sum_{i=1}^{n_{\text{obs}}} \left( -\eta \rho_i^k v_{RO} \vec{n}_{RO} \right)
\]

(74)

The derivation of these components are given in Appendix A.3. The sum all the attractive en repulsive forces is considered as the desired direction of motion.

### 6.2 Construction and application of the inverse Hessian function \( H^{-1} \)

To compute the Hessian function it is separated into an attractive part and a repulsive part as given in equation (75), where \( n_{\text{obs}} \) is the number of obstacles present in the field and \( U_{\text{rep}, i} \) is the potential function of obstacle \( i \).

\[
H = H(U_{\text{att}}) + H(U_{\text{rep}}) = \begin{bmatrix}
\frac{\partial^2 U_{\text{att}}}{\partial p_R^2} & \frac{\partial^2 U_{\text{att}}}{\partial p_R^2} \\
\frac{\partial^2 U_{\text{att}}}{\partial v_R^2} & \frac{\partial^2 U_{\text{att}}}{\partial v_R^2}
\end{bmatrix} + \sum_{i=1}^{n_{\text{obs}}} \begin{bmatrix}
\frac{\partial^2 U_{\text{rep}, i}}{\partial p_R^2} & \frac{\partial^2 U_{\text{rep}, i}}{\partial p_R^2} \\
\frac{\partial^2 U_{\text{rep}, i}}{\partial v_R^2} & \frac{\partial^2 U_{\text{rep}, i}}{\partial v_R^2}
\end{bmatrix}
\]

(75)

For the inverse of the Hessian to exist it needs to be positive definite. So before the inverse is calculated the determinant is checked to be positive, if not the Hessian is modified using the solution of Papalambros and Wilde (2000) given by equations (76) and (77). Solving equation (77) will result in two values for \( \mu \) from which the positive value will be used to correct the Hessian function.

\[
H_{\text{corrected}} = (H + \mu I) = \begin{bmatrix}
a + \mu & b \\
c & d + \mu
\end{bmatrix}
\]

(76)

\[
det(H_{\text{corrected}}) = \mu^2 + \mu(a + d) + (a \cdot d - b \cdot c) = 0
\]

(77)

Since the artificial forces are calculated locally it is preferred to also have a locally calculated Hessian. Otherwise the Hessian would contain components which originate from obstacles which are not contributed for in the artificial forces. This seems straightforward; however the implementation of the Hessian function has a great influence on the system behaviour as explained in the following example.

Consider an environment where no obstacles are present, i.e. the repulsive potential \( U_{\text{rep}} \) is zero. Then the desired direction of motion can be written as (78);

\[
\vec{F} = \sum \begin{bmatrix}
\frac{\partial^2 U_{\text{att}}}{\partial p_R^2} & \frac{\partial^2 U_{\text{att}}}{\partial p_R^2} \\
\frac{\partial^2 U_{\text{att}}}{\partial v_R^2} & \frac{\partial^2 U_{\text{att}}}{\partial v_R^2}
\end{bmatrix}^{-1} \cdot \begin{bmatrix}
\frac{\partial U_{\text{att}}}{\partial p_R} \\
\frac{\partial U_{\text{att}}}{\partial v_R}
\end{bmatrix}
\]

(78)

Using definition (73) for the attractive potential function \( U_{\text{att}} \), (78) can be written as:

\[
\vec{F} = \sum \begin{bmatrix}
2 \alpha_p & 0 \\
0 & 2 \alpha_v
\end{bmatrix}^{-1} \cdot \begin{bmatrix}
\alpha_p \rho_T \vec{n}_{RT} \\
\alpha_v v_{RT} \vec{n}_{VRT}
\end{bmatrix} = \frac{1}{2} \left( \rho_T \vec{n}_{RT} + v_{RT} \vec{n}_{VRT} \right)
\]

(79)
(79) no longer contains the parameters $\alpha_p$ and $\alpha_v$ which is a problem since the ratio between $\alpha_p$ and $\alpha_v$ determine the damping ratio $\xi$ as given by equation (62). If the Hessian is applied on the attractive artificial forces, the target approach phase is always damped for every non zero value of $\alpha_v$ since $\vec{n}_{RT}$ and $\vec{n}_{VRT}$ point in opposite directions (for a stationary target) and $p_T$ and $v_{RT}$ are of equal order. When obstacles are present, $U_{rep}/= 0$, within the proximity of the robot, the $H(1,2)$ and $H(2,1)$ components are non zero and the parameters $\alpha_p$ and $\alpha_v$ still occur in the artificial force $\vec{F}$. However, in this case it depends purely on the configuration between obstacle, robot and target for which ratio of $\alpha_p$ and $\alpha_v$ the system behaviour is damped or not.

To solve this problem only the repulsive artificial forces $\vec{F}_{rep,p}$ and $F_{rep,v}$ are modified with the inverse Hessian function. Doing so the desired direction of motion becomes:

$$\vec{F} = \vec{F}_{att,p} + \vec{F}_{att,v} + \sum \left( \sum_{i=1}^{n_{obs}} \begin{bmatrix} \frac{\partial^2 u_{att}}{\partial p_R^2} & \frac{\partial^2 u_{att}}{\partial p_R \partial v_R} \\ \frac{\partial^2 u_{att}}{\partial v_R \partial p_R} & \frac{\partial^2 u_{att}}{\partial v_R^2} \end{bmatrix} \cdot \begin{bmatrix} F_{rep,p} \\ F_{rep,v} \end{bmatrix} \right)$$

(80)

This action is justified since $H(U_{att})$ only changes the size of the vectors $\vec{F}_{att,v}$ and $\vec{F}_{att,p}$ and not the direction. Implementing (80) results in a controlled system behaviour, i.e. equation (62) holds again and the system can be set to be not damped, under-damped, critically damped or over-damped.

Figure 6.1 shows the effect of the Hessian modification. The Hessian components are a measure of the curvature of the potential field $U$ in a certain direction. If the curvature is much larger in the $\vec{F}_{rep,v}$ direction than in the $\vec{F}_{rep,p}$ the total repulsive force $\vec{F}_{rep}$ will be shifted towards the position component $\vec{F}_{rep,p}$. Effectively, steering the robot away from highly curved area in the potential field $U$.

![Figure 6.1: Modification of the artificial force $\vec{F}$ with the inverse Hessian.](image)

**6.3 Addition of the escape force $\vec{F}_e$**

A local minimum is detected if the following three conditions are satisfied. Conditions (81) and (82) are explained in section 4.2 and originate from the solution of Vadakkepatt and Tan (2000) for escaping local minima. Criteria (83) is added to assure the escape force does not drive the robot away from the target position.
If a local minimum is present, the escape force $\vec{F}_e$, given by (84), is added to the artificial force $\vec{F}$.

$$\vec{F}_e = \alpha_e \frac{1}{(\rho_s - \rho_r - \rho_m)^2} \vec{n}_{rep} \perp $$

(84) 

Where; $\vec{n}_{rep} \perp$ is a unit vector perpendicular to the repulsive force $\vec{F}_{rep}$ and $\alpha_e = 10^2 \cdot z = 10 \log (\vec{F}_{rep})$. The values of $\rho_s$ and $\rho_m$ are based upon the position and velocity information of the closest obstacle.

Before the escape force is added to the artificial force, the angle between the velocity vector $\vec{v}_e$ and the unit vector $\vec{n}_{rep} \perp$ is checked to be less or equal to $\frac{1}{2} \pi$. If not $\vec{n}_{rep} \perp$ is defined as $-\vec{n}_{rep} \perp$. This prevents the escape force from slowing the robot down and allows for a faster escape from a local minimum, since robot can continue in the direction it is already travelling. A second check prevents the escape force from pushing the robot over the space boundary; if the robot is within a certain distance from a boundary, $\vec{F}_e$ never points towards that boundary.

6.4 Gradient tracing

In order to correctly trace the field gradient the sliding motion control of Utkin and Guldner (2009) is implemented. Since no information is known about the type of robot, a unity matrix is used for the inertia matrix $M$ and the body forces $N$ are neglected. An arbitrarily robot mass of 1 kg is chosen, maximum acceleration is set to $a_{max} = 5 \text{ m/s}^2$, the maximum velocity is $v_{max} = 4 \frac{\text{ m}}{\text{ s}}$ and the maximum available control force is $\tau_0 = 5 \text{ N}$. 

$$\frac{\vec{F}_{total}}{\sum_{i=1}^{N_f} \vec{F}_{rep,i}} < 0.3$$

(81) 

$$\cos(\angle \vec{F}_{att} - \angle \sum_{i=1}^{N_o} \vec{F}_{rep,i}) < -\cos(\frac{\pi}{6})$$

(82) 

$$\|\vec{p}_{car} - \vec{p}_R\| > 0.5$$

(83)
7. Simulation results: System behaviour.

To determine the system behaviour some simple Target-Obstacle configurations are simulated. Special attention is put on the velocity of the robot $v_R$, acceleration $a_R$, path and distances $\rho_o$ and $\rho_T$. In the figures below; the green dot represents the target (or black star); obstacles are represented by the two red circles, where the inner circle is the sum of the robot and obstacle radii and the outer dashed circle is the influence zone $\rho_o$. The blue dot stands for the robot, the vectors pointing from it correspond to the artificial forces scaled to unit length. The green and red arrows are the attractive en repulsive forces, $\vec{F}_{att}$ and $\vec{F}_{rep}$ respectively. The black vector is the escape force $\vec{F}_e$ and the blue vector is the sum of all artificial forces $\vec{F}$.

Remark: As will be shown in section 7.4 the definition of the Hessian function $H$ as explained in section 6.2 does not improve system behaviour therefore it is only used in the simulations shown in section 7.4.

7.1 Desired behaviour

The main objective of the algorithm is to efficiently avoid obstacles while navigating towards the target position. It is desirable that the robot follows a smooth and short path with no oscillations to ensure a short assignment completion time $t_{com}$. During navigation two phases can be distinguished: the obstacle avoidance phase and the target approach phase. While avoiding obstacles it is sought after to have a high robot velocity $v_R$ at a safe distance from obstacles. When approaching the target it depends on the type of application if a hard or soft landing approach is required. To accommodate for both options it is desirable to have a system which can be tuned to be over-damped, critically damped or under-damped.

7.2 Obstacle avoidance phase

The first simulation configuration consists of an obstacle which is placed directly in front of the robot. Figure 7.1a illustrates the path which the robot followed to reach the target. Although the path generated seems quite smooth, the velocity and acceleration profiles, Figures 7.2(a, b) show some discontinuities.

Figure 7.2(a) illustrates the velocity profile experienced during navigation, from $t[\text{step}]=70$ the target approach phase starts, the robot velocity is lowered by using the solution of Utkin and Guldner (2009). Since the weigh factor $\alpha_\nu$ is set to an arbitrary value of zero the system is undamped which explains the oscillations around the target position from $t[\text{step}]=85$. These oscillations can be reduced using a non zero value for $\alpha_\nu$. A more detailed description on how to control the target approach is given in section 7.3.

Looking at the obstacle avoidance phase, the first point of interest is $t[\text{step}]=16$, at this point the acceleration profile shows a jump from maximum acceleration to maximum deceleration which results in a dip in the velocity.
profile. The cause of this jump is shown in Figure 7.3(a,b). At $t[\text{step}] = 15$ the artificial force $\vec{F}$ (blue vector) points towards the obstacle, i.e. the attractive force $\vec{F}_{\text{att}}$ is much larger than the repulsive force $\vec{F}_{\text{rep}}$, one time step later the repulsive force $\vec{F}_{\text{rep}}$ is equally sized to the attractive force $\vec{F}_{\text{att}}$ which results in a sudden change in direction of the total artificial force $\vec{F}$. Since the implemented solution of Utkin and Guldner (2009) is based upon reducing the error between the current velocity of the robot $\vec{v}_R$ and the desired velocity $\vec{v}_{\text{des}}$ a maximum deceleration is applied to reduce this error. The change in direction seems like a discrete event but this is not the case. The angle over which the desired direction of motion changes within one time step can be reduced by choosing smaller iteration steps. However since the acceleration available to trace the field gradient is limited a dip in the velocity profile will always be present, i.e. a smaller time step also results in less time for the robot the reduce the velocity error between $\vec{v}_R$ and $\vec{v}_{\text{des}}$. A combination of a higher maximum acceleration and smaller time steps allows for better tracing of the field gradient which will eliminate the dip.

At $t[\text{step}] = 32$ a similar situation is present though with a different cause; as seen in Figure 7.4(a,b) the artificial force $\vec{F}$ changes over a significant angle within one step. The cause of this change can be seen in Figure 7.1(b); at $t[\text{step}] = 32$ the robot is moving towards the robot, at $t[\text{step}] = 33$ the robot is moving away from the robot, i.e. the relative velocity between robot and obstacle $v_{\text{ROI}}$ turns from negative to positive. Since the repulsive potential is defined to be zero if $v_{\text{ROI}} > 0$, the repulsive forces start to decay from this point. By implementing a less abrupt decay of the repulsive forces $\vec{F}_{\text{rep}}$ and increasing the influence area of the decaying force from the point where the criteria $\rho_s = \rho_0$ is met to the point where the criteria $\rho_s - \rho_R - \rho_m = \rho_0$ is met, the bump can be eliminated as shown in Figure 7.5(a,b); (a) gives the velocity profile where the decaying force is defined as $\vec{F}_{\text{dec}} = 0.9^n\vec{F}_{\text{rep}}$. 
where $n$ is the number of time steps taken from the moment $v_{ROI} > 0$ until the robot is no longer within the influence zone $\rho_0$; (b) illustrates the resulting velocity profile when $\tilde{F}_{\text{dec}} = 0.2^n \tilde{F}_{\text{rep}}$ is used. The price paid for the elimination of the bump is shown in Figure 7.6; after passing the obstacle the robot is still influenced by the obstacle which results in a detour and a longer path of motion towards the target. It might even drive the robot towards other obstacles.
7.2.1 Conclusion
The algorithm can effectively avoid obstacles and generate a smooth path towards the target. However, the velocity and acceleration profiles show some abrupt changes. Some of these changes can be eliminated by choosing a smaller iteration step size and provide a larger maximum acceleration for the robot which enables the robot to trace the field gradient more accurately. A second option is to provide a smoother transition between the repulsive force $f_{d_{re}}$ and the decay force $f_{d_{dc}}$, although the increased path length is a undesirable side effect.

7.3 Target approach phase
To investigate the behaviour of the system while the robot is approaching the target; no obstacle is placed between the robot and the target, so the robot is allowed to freely move towards the target. The algorithm contains two methods which can slow down the robot near the target and provide a soft or hard landing approach. The first solution uses the modified potential field method of GE and Cui (2002). By tuning the parameters $\alpha_p$ and $\alpha_v$, the system can de set to be undamped, under-damped, critically damped and over-damped as shown in the Figures 7.6 to 7.9.

If $\alpha_v = 0$, the system is undamped and will not slow down near the target, instead it will oscillate around the target position in its Eigen-frequency, as seen in Figure 7.6. If $\alpha_v > 0$, oscillations will be damped and the robot will converge towards the target position. In Figure 7.7 the system is under damped causing the robot to overshoot the target and asymptotically approaching the target.

Remark: The target approach could also be tuned by modifying the parameters $m$ and $n$ as explained in Appendix A.2, however this is not simulated in this paper.
If the system is critically or over-damped the target will not overshoot the target but will approach it asymptotically. Although it must be kept in mind that a critically damped system is easily disturbed by nearby obstacles. The higher the damping ration $\xi$, the slower the robot will approach the target and the longer the task completion time will be. A variable $\xi$ could be a solution for this problem, however no option to vary $\xi$ is present in the algorithm. Implementing of variable parameter settings during navigation is an interesting option for further work. The oscillations seen in the velocity profile of the under-, critically- and over damped approaches are again caused by a sudden change in the desired direction of motion, as seen in figure 7.11.
At $t[step] = 62$ the desired direction of motion points away from the target, i.e. the velocity component of the attractive potential field $\vec{F}_{a,p}$ over powers the position component $\vec{F}_{a,p}$, one time step later it is the other way around. In this case a decrease in step size does not eliminate the sudden changes in direction, since the position component of the attractive force points in the direction of $\vec{n}_{RT}$ while the velocity component points in the $\vec{n}_{VRT} = -\vec{n}_{RT}$ direction, where $\vec{n}_{RT}$ is a unit vector pointing from the robot towards target. So the resulting force points in the direction of either the position component or the velocity component causing oscillations. This effect might be reduced by using the length of the artificial force vector in the calculation of the desired direction, i.e. a small $\|\vec{F}\|$ results in small acceleration. In the solution of Utkin and Guldner (2009) only the direction of the artificial force is considered as the most promising direction of motion which results in the steps from maximum acceleration to maximum deceleration as seen in Figure 7.2(b).

The second option used to control the target approach uses the solution of Utkin and Guldner (2009), which lowers the desired velocity as a function of the distance between robot and target $p_{RT}$. As seen in Figure 7.11 the position error between robot and target is reduced just as fast as in a critically damped system. However the oscillations in the velocity profile of a critically damped system have a much smaller amplitude than when the desired velocity $v_{des}$ is lowered near the target position.

The solution of Utkin and Guldner (2009) can also be combined with slightly over-damped parameter settings as shown in Figure 7.12(a,b). This results in a fast approach with fewer oscillations around the target position. Moreover, damped parameter settings will increase system stability if obstacles are near the target. So this combination of lowering the maximum velocity near the target and damped parameter settings is considered as the preferred setting for the new algorithm.
7.3.1 Conclusion
As seen in the figures above the system can be tuned to be under-damped, critically damped or over-damped. A combination of lowering the maximum velocity near the target and slightly damped parameter settings results in a fast approach while reducing oscillations around the target position. In future implementation variable parameter settings are recommended to reduce oscillations and increase system stability. A second recommendation is to use the length of the artificial force factor in the computation of the desired acceleration, i.e. a small $\|F\|$ should result in a small acceleration.

7.4 Influence of the inverse Hessian function
Ren and McIsaac (2006) proved the Hessian function effectively reduces oscillations in close proximity of obstacles or narrow passages. However, in this paper the Hessian is no longer a function of $x$ and $y$ but of $\vec{p}_R$ and $\vec{v}_R$. So to check if application of the Hessian function still has a positive effect a simulation study is done.

Figure 7.13 illustrates the space configuration used in this simulation: four obstacles are placed between the robot and the target. 7.13(a) shows the path followed by the robot with modification of the repulsive forces with the Hessian function; in 7.13(b) the path followed by the robot without modified repulsive forces. Looking at Figure 7.13 and 7.14 it can be seen that the robot follows a smoother path without modification of the repulsive forces. Moreover, without modification the assignment completion time is about 100 steps ($= 3.13 \text{ s}$), with modifications it takes about 140 steps ($= 4.38 \text{ s}$). The velocity profiles in Figure 7.16 show even bigger differences. Against expectations, the application of the inverse Hessian function does not reduce oscillations and does not provide faster progress.

![Figure 7.13(a,b): Path travelled with Hessian(a); Path travelled without Hessian(b)](image)

![Figure 7.14(a,b): Distances from robot to target and obstacle, with (a) and without Hessian function(b)](image)
Figure 7.15(a,b): Velocity profile with Hessian(a); Velocity profile without Hessian(b)

The reason for this ‘malfunctioning’ of the Hessian function can be found in the components of the inverse Hessian $H^{-1}$. The $H^{-1}(2,2)$ component is consistently much larger than the $H^{-1}(1,1)$ component which implies that the modified repulsive force $H^{-1} \cdot \vec{F}_{\text{rep}}$ is strongly shifted towards the velocity component of the repulsive force $\vec{F}_{\text{rep}}$. Figure 7.16 illustrates this problem; the blue vector is the position component of the repulsive force $\vec{F}_{\text{rep},p}$; the pink vector represents the velocity component $\vec{F}_{\text{rep},v}$; the red vector is the resulting repulsive force $\vec{F}_{\text{rep}}$ and the yellow vector is the repulsive force after modification with the inverse hessian function $H^{-1} \cdot \vec{F}_{\text{rep}}$. Figure 7.16 clearly shows the strong rotation towards the direction of the velocity component.

Looking at the definitions of the position and velocity components of the repulsive force; given by equations (85) and (86):

\[
\vec{F}_{\text{rep},p} = \frac{-\eta \rho_F^k}{(\rho_s - \rho_R - \rho_m(v_R))} \vec{n}_{RO} + \frac{\eta v_{RO} \rho_{RO} \rho_F^k}{\rho_s a_{\text{max}}(\rho_s - \rho_R - \rho_m(v_R))^2} \vec{n}_{RO} + k \eta \left( \frac{1}{\rho_s - \rho_R - \rho_m(v_R)} - \frac{1}{\rho_{RO}} \right) \rho_F^{-1} \vec{n}_{RT} \tag{85}
\]

\[
\vec{F}_{\text{rep},v} = \frac{-\eta \rho_F^k v_{RO}}{a_{\text{max}}(\rho_s - \rho_R - \rho_m(v_R))^2} \vec{n}_{RO} \tag{86}
\]
It is seen that $\vec{F}_{\text{rep},v}$ only contains a component in the direction of $\vec{n}_{RO}$ while $\vec{F}_{\text{steep}}$ contains the ‘steering’ components in the directions of $\vec{n}_{RO,\perp}$ and $\vec{n}_{RT}$. As a result of the Hessian function the ‘breaking’ action in the direction of $\vec{n}_{RO}$ is enhanced while the ‘steering’ action is reduced which explains the slow progress and reduced velocity nearby obstacles.

**7.4.1 Conclusion**
As shown in the figures above application of the inverse Hessian function, defined as a function of $\vec{p}_R$ and $\vec{v}_R$, does not improve system behaviour. The inverse Hessian rotates the direction of the repulsive force $\vec{F}_{\text{rep}}$ towards the direction of the velocity component $\vec{F}_{\text{rep},v}$. As a result ‘breaking’ action in the direction of $\vec{n}_{RO}$ is enhanced while steering action in the direction of $\vec{n}_{RO,\perp}$ and $\vec{n}_{RT}$ is reduced, this in turn results in slow progress.

**7.5 Escaping local minima**
As explained before escaping local minima is an important aspect of robot motion planning algorithms based upon the potential field method. Figure 7.17(a) shows that the robot can effectively escape a local minimum using the escape force; 7.17(b) shows the attractive and repulsive force (red and green vectors) aligned opposite of each other, the escape force denoted with the black vector is applied perpendicular to the repulsive force guiding the robot past the obstacles.

![Figure 7.17: Path followed to escape local minima](image)

![Figure 7.18: Velocity profile](image)

The velocity profile shows some oscillations between $t[\text{step}] = 15$ and $t[\text{step}] = 25$, these oscillations are caused by the sudden change in direction of the desired direction of motion when the escape force is turned on and turned off. However this is a small price to pay if the robot can effectively escape local minima.
7.6 Dynamical obstacles
The main reason for using the velocity information of robot, target and obstacles into the computation of the potential field $U$ was to enable the robot to avoid dynamical obstacles. Several simulations are run to test the dynamical obstacle avoidance capacity of the algorithm. Figure 7.19 illustrates the path followed by the robot while avoiding a head-on collision at maximum velocity. The Figures 7.20 to 7.21 show obstacles travelling at maximum velocity from right to left; from right lower corner to left upper corner and from bottom to top. In each situation the robot was able to successfully avoid the obstacle and reach the target. The only case in which the robot was not always able to avoid an obstacle is shown in Figures 7.23 and 7.24; when the obstacle is travelling in the same direction as the robot and the target approach phase starts, the robot decelerates which causes a collision. However when the robot is not near the target the obstacle can be successfully avoided.

Figure 7.19: Avoiding an obstacle travelling at maximum velocity from right upper corner towards left lower corner

Figure 7.20: Avoiding an obstacle travelling from right to left at maximum velocity
Figure 7.21: Avoiding an obstacle travelling at maximum velocity from right lower corner towards left upper corner

Figure 7.22: Avoiding an obstacle travelling at maximum velocity from bottom to top.

Figure 7.23: Avoiding an obstacle which is travelling in the same direction as the robot.
Figure 7.24: A collision occurs when a robot is chasing the robot with maximum velocity and the robot is near the target.

From Figures 7.19 to 7.24 it can be concluded that in most cases the algorithm can successfully avoid dynamical obstacles, accept when the robot is chased by the obstacle and is near the target.

### 7.7 Dynamical Target
The second reason for using the velocity information into the computation of the potential field $U$ was to enable the robot to trace a dynamical target. In Figure 7.25 the target is travelling from the left upper angle towards the right lower angle (black line). The path followed by the robot depends on the settings of the parameters $\alpha_p$ and $\alpha_v$.

![Figure 7.25](image)

**Figure 7.25:** Approaching target with $\alpha_p = 10$, $\alpha_v = \sqrt{20}$ (a) and $\alpha_p = 10$, $\alpha_v = 2 \cdot \sqrt{20}$ (b)

### 7.8 Space Boundaries
The robot does not always stay within the space boundaries, especially when the robot is travelling near maximum velocity. This problem is also present in Mouton, placing a ghost obstacle at the wall works in some cases but not in all.

**Remark:** The algorithm is not always able to generate a safe, collision free path towards the target. This is caused by the fact that system behaviour is highly depended on the ratio between the parameters $\alpha_p$, $\alpha_v$, $\alpha_e$ and $\eta$ which are the weigh factors for the position component of the attractive artificial force $\vec{F}_{att.p}$; the velocity component of the attractive artificial force $\vec{F}_{att.v}$, the escape force $\vec{F}_e$ and the repulsive forces $\vec{F}_{rep}$ respectively. For future implementation it is recommended to use on-line variable parameters.

40
8. Simulation results: Performance comparison

The goal of this report was to compare the previously develop algorithm of Mouton with an algorithm based upon potential field literature. In this chapter several obstacle target configurations are used to compare both algorithms regarding task completion time; length of path travelled and smoothness of the velocity profile. In order to make a fair comparison the new algorithm is set to be an undamped system, and the maximum allowable velocity is not lowered near the target position, i.e. the robot is allowed to approach the target at maximum velocity as is in L.Mouton. Both systems are undamped, so both will oscillate around the target position at their natural frequencies. Since obstacle avoidance and path finding is the most important aspect of the algorithms; the assignment is said to be completed when the robot is within a distance of 0.1 m of the target.

8.1 One obstacle

In the first simulation an obstacle is placed directly in-between the obstacle and the target. Since the L.Mouton algorithm cannot avoid an obstacle which is placed exactly between target and robot, the obstacle it is shifted 0.05 m from the robot-target line. Figures 8.1 and 8.2 give the paths and velocity profiles of the new algorithm and of L.Mouton. The new algorithm generates a slightly longer path length towards the target, however the average velocity is 0.5 m/s higher which results in a shorter task completion time.

![Figure 8.1: Path and velocity profile of new algorithm.](image1)

![Figure 8.2: Path and velocity profile Mouton](image2)
The resulting velocity profile of the new algorithm shows fewer discontinuities than the L.Mouton algorithm, which is a result of the fact that the maximum allowed velocity is not changed near obstacles and the repulsive force decays more gradually.

8.2 Local minima
In this section the ‘escaping local minima’ capacities of both algorithms are compared. Two simulations are run; the first simulation uses two obstacles to create a local minimum; the second uses three obstacles to form a local minimum.

8.2.1 Two Obstacles
Figures 8.3 and 8.4 clearly illustrate the benefit of using the escape force method of Vadakkepat and Tan (2000). Before the robot enters the local minimum it already starts an avoiding action which results in a fast task completion time. The Mouton robot first enters the local minimum and has to wait until a virtual obstacle is created which pushes the robot into a direction from which it can continue towards the target. Moreover, the creation of the virtual obstacles results in strong fluctuations in the velocity profile; while the implementation of the escape force only results in two slight discontinuities at $t[\text{step}] = 18$ and $t[\text{step}] = 32$.

Figure 8.3: Path and velocity profile of new algorithm.

Figure 8.4: Path and velocity profile Mouton.
8.2.2 Three obstacles

When three obstacles are used to form a local minimum it is not always possible to avoid the minimum using the escape force. It depends on the weigh factor of the escape force whether or not an escape is possible. Figure 8.5 shows the failed attempt to escape the local minimum; a slight curve (between 1-1.5) is seen in the robot path where the escape force tried to pull the robot away from the minimum. When the weighing factor $\alpha_v$ is increased from $10^2$ to $100^2$ where $z = 10 \log (\rho_{rep})$ the escape force is sufficiently large enough to drive the robot away from the minimum, as shown in Figure 8.6. A variable weighing factor might be the solution to ensure the robot is always able to avoid a local minimum.

The solution of Mouton (Figure 8.7) also enables the robot to escape a local minimum formed by three obstacles, although task completion time is over four times as long. So avoiding a local minimum using the escape force is preferred but the solution of Mouton could be used as an extra safe guard for when the escape force is not sufficient.
8.3 Dynamical Environment
The sections 7.6 and 7.7 proved the new algorithm enables the robot to avoid dynamical obstacles and trace a moving target. Since L.Mouton could not avoid a head on collision and could not efficiently intercept a moving target, the introduction of the velocity information into the potential field is considered as desirable. Moreover, the usage of the velocity information allows for a controlled (damped) target approach which is not possible in the Mouton algorithm since it is always under-damped.

8.4 Target within influence zone of robot
The L.Mouton algorithm is not able to safely approach a target which is near to an obstacle as shown in section 5.1. Using the modified repulsive force of GE and Cui (2000) this is not a problem anymore as shown in Figure 8.9.

8.5 Conclusion
The new developed algorithm has a consistently shorter task completion time; generates a smoother velocity profile; can safely reach a target near an obstacle; can easily avoid a head on collision at maximum velocity and can trace a dynamic target which previously was not possible. In most cases the implemented escape force allows for a quick escape from local minima. In situations where the escape force is insufficient a variable weigh factor might enhance performance or the virtual obstacle placement algorithm could be used as a back-up system.
9. Conclusion

The basic concept behind the newly developed algorithm and the Mouton algorithm is the same: the robot should be attracted by the target and should be repulsed by obstacles. To establish these attractive and repulsive properties both algorithms use a potential field $U$ in which the target is represented by a global minimum and the obstacles correspond to local maxima. The main difference between the two algorithms lies within the definition of the potential field $U$. Mouton considers the potential field $U$ only to be a function of the position $\mathbf{p}_R$ of the robot, while in the new algorithm the potential field $U$ is defined as a function of position $\mathbf{p}_R$ and velocity $\mathbf{v}_R$ of the robot. The reason for implementing the velocity information into the potential field $U$ is to enhance avoidance of dynamical obstacles and enable the robot to trace a dynamical target.

In both algorithms the desired direction of motion is calculated by locally differentiating the constructed potential field $U$ which results in a so called artificial force $\mathbf{F} = -\nabla U$, where Mouton differentiates towards $x$ and $y$ and the new algorithm towards $\mathbf{p}_R$ and $\mathbf{v}_R$. Mouton modifies the direction of the artificial force $\mathbf{F}$ by multiplying it with the inverse Hessian function $H^{-1}$ which contains all second order derivatives of the potential field $U$ towards $x$ and $y$; the resulting direction of $H^{-1} \cdot \mathbf{F}$ is named the Newton direction and is considered as the most promising direction of motion. Ren and McIsaac (2006) proved usage of the inverse Hessian function enhances system performance since a less steep route is generated. Given that desirable effect, a Hessian function is derived for the new algorithm which contains all the second order derivatives towards $\mathbf{p}_R$ and $\mathbf{v}_R$. However, this definition of the Hessian function does not improve system performance. Since the desired direction of motion is always rotated towards the direction of the velocity component of the artificial force $\mathbf{F}$. This only causes the robot to decelerate.

As a consequence the Hessian function has not been implemented into the algorithm and the direction of the artificial force $\mathbf{F}$ is considered as the most promising direction of motion.

When the desired direction of motion is calculated both algorithms are designed to reduce the error between the current velocity of the robot $\mathbf{v}_R$ and the desired velocity direction $\mathbf{v}_{des}$. Acceleration and velocity is limited in the new algorithm as well as in Mouton. However, the new algorithm allows for the implementation of the dynamical model of the robot since it uses the inertia mass matrix $M$ and available control forces $\mathbf{T}$ to calculate the applied acceleration. This ensures a more realistic behaviour of the robot during simulation.

While travelling the robot could encounter local minima. To escape from those minima the new algorithm uses an escape force which pulls the robot away from the minimum position. Mouton placed virtual obstacles at minima positions to push the robot away from that location. In most cases the implemented escape force allows for a quick escape from local minima which results in a much shorter task completion time. However, in some situations the escape force is insufficient, for those occasions the virtual obstacle placement algorithm could be used as a back-up system.

Simulation studies showed that the newly developed algorithm has a consistently shorter task completion time; generates a smoother velocity profile; can safely reach a target near an obstacle; can easily avoid a head on collision at maximum velocity and can trace a dynamic target which previously was not possible. Moreover, the target approach phase can be controlled to be undamped, damped or over-damped due to the usage of the velocity information in the formulation of the potential field $U$.

9.1 Recommendations for future implementation

Despite system performance is improved compared to Mouton, it can not be guaranteed that the algorithm will always find a safe, collision free path to reach the target. Since system behaviour is highly depended on the ratio between the parameters $\alpha_p$, $\alpha_v$, $\alpha_e$ and $\eta$ which are the weigh factors for the position component of the attractive artificial force $\mathbf{F}_{att,p}$; the velocity component of the attractive artificial force $\mathbf{F}_{att,v}$; the escape force $\mathbf{F}_e$ and the repulsive forces $\mathbf{F}_{rep}$ respectively. In some situations a strong attractive force is required to pull the robot through narrow passages, however in other situation a too strong attractive force might result in a collision. For future implementation it is recommended to use variable weigh factors to improve system performance.
The solution of Utkin and Guldner (2009) used to converge towards the field gradient only considers the direction of the artificial force $\vec{F}$ but not the actual length of the vector $\|\vec{F}\|$. As a result, the acceleration profile of the robot often shows discontinuous steps from maximum acceleration to maximum deceleration. Using the length of the artificial force factor in the computation of the desired acceleration, i.e. a small $\|\vec{F}\|$ should result in a small acceleration, might solve this problem.

A problem that still exists within the new algorithm is that of space boundaries; the robot does not always stay within the space boundaries, especially when the robot is travelling near maximum velocity. This problem is also present in Mouton, placing a ghost obstacle at the wall works in some cases but not in all. The usage of the point array method could result in better performance. On top of that, the point array method also accommodates for the modelling of non-circular objects which is not possible in the current algorithm. Another unresolved issue is the functioning of the Hessian function. A different definition should be used in order to apply the Hessian correctly.
References

- Utkin V., Guldner J. and Shi J. (2009), ‘Sliding mode control in electro mechanical systems’, Taylor and Francis Group
Appendix A.1: The Hessian function.

The Hessian function given by equation (A.1) contains all second order partial derivatives of a given function $U(\vec{p}_R)$ which is a function of $x$ and $y$, i.e. $\vec{p}_R = f(x, y)$

\[
H = \begin{bmatrix}
\frac{\partial^2 U(\vec{p}_R)}{\partial x^2} & \frac{\partial^2 U(\vec{p}_R)}{\partial x \partial y} \\
\frac{\partial^2 U(\vec{p}_R)}{\partial y \partial x} & \frac{\partial^2 U(\vec{p}_R)}{\partial y^2}
\end{bmatrix} \in \mathbb{R}^2 , \text{ Hessian function} \quad (A.1)
\]

Since the inverse of $H$ is used it needs to be positive definite, i.e. $\det(H) > 0$. If this is not the case, it can be modified by using the solution of Papalambros and Wilde (2000).

\[
H^* = H + \mu I \quad , \text{Positive definite scaled Hessian} \quad (A.2)
\]

\[
H^* = \begin{bmatrix}
\alpha + \mu & b \\
\frac{b}{d} & d + \mu
\end{bmatrix} \quad (A.3)
\]

\[
\det(H^*) = (\alpha + \mu) \cdot (d + \mu) - b \cdot c = \mu^2 + \mu(a + d) + (a \cdot d - b \cdot c) = 0 \quad (A.4)
\]

Solving equation (A.4) (using the quadratic-formula) results in two values of $\mu$ from which the largest positive value is chosen. A slightly large value is added ($1.01 \cdot \mu I$) to the Hessian in order to make it positive definite.

In this paper the potential function is both a function of position $\vec{p}_R$ and velocity $\vec{v}_R$. So the total potential function $U(\vec{p}_R, \vec{v}_R)$ is a function of two variables which results in a $2 \times 2$ Hessian matrix.

\[
H(\vec{p}_R, \vec{v}_R) = \begin{bmatrix}
\frac{\partial^2 U}{\partial \vec{p}^2}_R & \frac{\partial^2 U}{\partial \vec{p} \vec{v}}_R \\
\frac{\partial^2 U}{\partial \vec{v} \vec{p}}_R & \frac{\partial^2 U}{\partial \vec{v}^2}_R
\end{bmatrix} \in \mathbb{R}^2 \quad (A.5)
\]
Appendix A.2: The influence of $m$, $n$, $\alpha_p$ and $\alpha_v$ on the potential function.

The attractive potential function is given by equation (A.6). Since all the parameters are positive the system is stable, however different combination of them result in different behaviour of the system. Ge and Cui (2002) did numerical experiments to show the influence of the different constant parameters.

\[
U_{\text{attractive}}(\vec{p}, \vec{v}) = \alpha_p \| \vec{p}_{\text{targ}} - \vec{p}_R \|^m + \alpha_v \| \vec{v}_{\text{targ}} - \vec{v}_R \|^n
\]  

(A.6)

Where,

- $\vec{p}_{\text{targ}}$, $\vec{p}_R$ are the positions of the target and the robot
- $\vec{v}_{\text{targ}}$, $\vec{v}_R$ are the velocities of the target and robot

and,

- $\| \vec{p}_{\text{targ}} - \vec{p}_R \|$ is the Euclidean distance between robot and target
- $\| \vec{v}_{\text{targ}} - \vec{v}_R \|$ is the relative velocity between target and robot

- $\alpha_p$ and $\alpha_v$, are positive scalar parameters
- $m$ and $n$, are positive scalar parameters

For an omnidirectional robot the damping ratio and natural frequency are given by (A.7) and (A.8) respectively.

\[
\xi = \frac{\alpha_v}{\sqrt{2\alpha_p}}
\]  

(A.7)

\[
\omega_n = \sqrt{2\alpha_p}
\]  

(A.8)

Assume $m = n = 2$. In Figure A.2.1.a the approach path of the robot can be seen for different values of $\xi$. For the general potential function $\alpha_v = 0$ which leads to a zero damping ratio $\xi$. As a result of this the robot will oscillate around its target position in its natural frequency $\omega_n$. Introducing the velocity information into the target function, results in a non-zero damping ratio which ensures the position error will reduce asymptotically. For $0 < \xi < 1$ the system is under-damped which means the tracing error will reduce with oscillation. If $\xi > 1$, the system is overdamped, no overshoot will occur but approach is also slow. In Figure A.2.1.b $\xi$ is held constant but the values of $\alpha_p$ and $\alpha_v$ are varied. By tuning these parameters different system behaviour can be obtained.

Figure A.2.2a,b show the influence of $m$ and $n$. If now $\xi$ and $n = 2$ are held constant and $m$ is varied from 1.5 to 3, it can be seen that the bigger the parameter, the faster the position error between the robot and target decreases. Although, $m = 3$ causes the robot to overshoot its target. The other way around, if $\xi$ and $m = 2$ are held constant and $n$ is varied from 1.5 to 3, it can be seen that the larger $n$ the faster position and velocity errors reduce to zero. In this case, $n = 3$ causes the robot to oscillate around the target. (Figure A.2.2a,b)
Figure A.2.1.a,b: Paths for robot and target for different values of $\xi$; Paths for robot and target for different combinations of $\alpha_p$ and $\alpha_v$. (Ge and Cui (2002))

Figure A.2.2.a,b: Paths of robot and target for varying $m$ and $n$ respectively. (Ge and Cui (2002))
Appendix A.3: The new repulsive Force

In order to correctly implement the sections 3 to 3.2 into the algorithm a new formulation for the repulsive force has to be derived. This derivation results in the repulsive force of one obstacle.

$$U_{rep}(\vec{p}_R, \vec{v}_R) = \eta \left( \frac{1}{\rho_s - \rho_R - \rho_m} - \frac{1}{\rho_0} \right) \rho_T^k$$  \hspace{1cm} (A.9)

Where,

- $\eta$, a positive constant
- $\rho_s = (\vec{p}_{obst,i} - \vec{p}_R) \cdot \vec{n}_{RO}$, relative distance between robot and target
- $\rho_R$, radius of robot
- $\rho_m = \frac{v_{RO}^2}{2 \sigma_{max}}$, distance travelled while relative velocity is reduced to zero
- $v_{RO} = (\vec{v}_{obst,i} - \vec{v}_R) \cdot \vec{n}_{RO}$, relative velocity between robot and obstacle in the direction from the robot towards the obstacle
- $\rho_0$, influence region of the obstacles
- $\rho_T^k = (\vec{p}_T - \vec{p}_R) \cdot \vec{n}_{RT}$, relative distance between robot and target
- $k$, positive constant

As said before the total repulsive force is defined as the negative gradient of the repulsive potential function (A.10)

$$\vec{F}_{rep}(\vec{p}_R, \vec{v}_R) = -\nabla U_{rep}(\vec{p}_R, \vec{v}_R)$$  \hspace{1cm} (A.10)

$$= -\frac{\partial U_{rep}(\vec{p}_R, \vec{v}_R)}{\partial \vec{p}_R} - \frac{\partial U_{rep}(\vec{p}_R, \vec{v}_R)}{\partial \vec{v}_R}$$

Where,

$$-\nabla_\vec{p} U_{rep} = \eta \left( \frac{v_{RO} + \rho_T^k}{\rho_0} \frac{1}{\rho_s - \rho_R - \rho_m} \frac{\rho_T^k}{\rho_T^k + k \eta} \frac{1}{\rho_s - \rho_R - \rho_m} \frac{1}{\rho_0} \right) \vec{n}_{RO} + k \eta \frac{1}{\rho_s - \rho_R - \rho_m} \frac{1}{\rho_0} \vec{n}_{RT} \hspace{1cm} (A.11)$$

$$-\nabla_\vec{v} U_{rep} = \frac{\eta v_{RO} \rho_T^k}{\rho_0 \rho_s \rho_s \rho_R - \rho_m^2} \vec{n}_{RO} \hspace{1cm} (A.12)$$

For (A.11) is made use of the properties (A.13) and (A.14) (Ge and Gui (2000))

$$\nabla_\vec{n} v_{RO} = -\vec{n}_{RO} \hspace{1cm} (A.13)$$

$$\nabla_\vec{p} v_{RO} = -\frac{1}{\|\vec{p}_R - \vec{p}_{obst}\|} \cdot v_{RO} \vec{n}_{RO} \hspace{1cm} (A.14)$$

Adding (A.11), (A.12) and (A.13) results in the total repulsive force $\vec{F}_{rep}$

$$\vec{F}_{rep} = \vec{F}_{rep,1} + \vec{F}_{rep,2} + \vec{F}_{rep,3} \hspace{1cm} (A.15)$$

Where,
\[ F_{\text{rep,1}} = -\eta \rho_T^k \left( \frac{\rho}{(\rho_s - \rho_R - \rho_m)^2} \left( 1 - \frac{\rho_{RO}}{\alpha_{max}} \right) \right) \tilde{\rho}_{RO} \]  
(A.16)

\[ F_{\text{rep,2}} = \frac{-\eta v_{RO} v_{ROL} \rho_T^k}{\rho_s \alpha_{max} (\rho_s - \rho_R - \rho_m)^2} \tilde{\rho}_{ROL} \]  
(A.17)

\[ F_{\text{rep,3}} = k\eta \left( \frac{1}{\rho_s - \rho_R - \rho_m} - \frac{1}{\rho_0} \right) \rho_T^{k-1} \tilde{\rho}_{RT} \]  
(A.18)

\[ = k\eta \left( \frac{1}{\rho_s - \rho_R - \rho_m} - \frac{1}{\rho_0} \right) \left( \frac{1}{\rho_T^{1+k}} \right) \tilde{\rho}_{RT} \]
Appendix A.4: Hessian components

In this appendix all components of the hessian function are analytically derived. The total potential function is given by equation (A.19)

\[ U(\vec{p}_R, \vec{v}_R) = U_{\text{att}}(\vec{p}_R, \vec{v}_R) + U_{\text{rep}}(\vec{p}_R, \vec{v}_R) \]  

\[ = \alpha_p \rho_R^m + \alpha_v v_{RT}^n + \eta \left( \frac{1}{\rho_R - \rho_-} \right)^{\frac{n}{2m}} \rho_R^k \]  

For this derivation the following properties are used. (Ge and Gui (2000))

\[ \nabla_{\vec{p}_R} \rho_s = -\vec{n}_{RO} \]  

(A.20)

\[ \nabla_{\vec{v}_R} \rho_s = 0 \]  

(A.21)

\[ \nabla_{\vec{p}_R} \rho_T = -\vec{n}_{RT} \]  

(A.22)

\[ \nabla_{\vec{v}_R} \rho_T = 0 \]  

(A.23)

\[ \nabla_{\vec{v}_R} v_{RO} = -\vec{n}_{RO} \]  

(A.24)

\[ \nabla_{\vec{p}_R} v_{RO} = \frac{1}{||\vec{p}_R - \vec{p}_{\text{obst}}||} \left[ v_{RO} \vec{n}_{RO} - (\vec{v}_R - \vec{v}_{\text{obst}}) \right] = -\frac{1}{||\vec{p}_R - \vec{p}_{\text{obst}}||} \cdot v_{RO} \vec{n}_{RO} \]  

(A.25)

\[ \nabla_{\vec{v}_R} v_{RO} = \vec{n}_{RO} \]  

(A.26)

\[ \nabla_{\vec{p}_R} v_{RO} = \frac{1}{||\vec{p}_R - \vec{p}_{\text{obst}}||} \left[ v_{RO} \vec{n}_{RO} - (\vec{v}_R - \vec{v}_{\text{obst}}) \right] = -\frac{1}{||\vec{p}_R - \vec{p}_{\text{obst}}||} \cdot v_{RO} \vec{n}_{RO} \]  

(A.27)

\[ \nabla_{\vec{v}_R} v_{RT} = -\vec{n}_{VRT} \]  

(A.28)

\[ \nabla_{\vec{p}_R} v_{RT} = 0 \]  

(A.29)

\[ \vec{n}_{RO\perp} \cdot \vec{n}_{RO} = 0 \]  

(A.32)

\[ \vec{n}_{RT\perp} \cdot \vec{n}_{RT} = 0 \]  

(A.33)

Differentiation: twice to position

\[ \frac{\delta U}{\delta \vec{p}_R} = \frac{\delta}{\delta \vec{p}_R} \left( U_{\text{att}} + U_{\text{rep}} \right) \]  

(A.34)

\[ \frac{\delta U_{\text{att}}}{\delta \vec{p}_R} = \frac{\delta}{\delta \vec{p}_R} \left( \alpha_p \rho_R^m + \alpha_v v_{RT}^n \right) \]  

\[ = \frac{\delta}{\delta \vec{p}_R} \left( m \alpha_p \rho_R^{m-1} \vec{n}_{RT} \right) \]  

\[ = (m - 1) m \alpha_p \rho_R^{m-2} \vec{n}_{RT} \]  

(A.35)
\[
\frac{\delta v_{\text{rep}}}{\delta \rho_R \rho_k} = \frac{\delta}{\delta \rho_R} \left( \eta \left( \frac{1}{\rho_s - \rho_R - \rho_m(v_{\text{ro}})} - \frac{1}{\rho_0} \right) \rho_T^k \right) - k\eta \left( \frac{1}{\rho_s - \rho_R - \rho_m(v_{\text{ro}})} - \frac{1}{\rho_0} \right) \rho_T^{k-1} \tilde{n}_{RT}
\]

\[
= \frac{\delta}{\delta \rho_R} \left( \eta \frac{\rho_T^k}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RO} - \eta v_{\text{ro}} v_{\text{ro}, \rho} \rho_T^k \tilde{n}_{RO} - k\eta \left( \frac{1}{\rho_s - \rho_R - \rho_m(v_{\text{ro}})} - \frac{1}{\rho_0} \right) \rho_T^{k-1} \tilde{n}_{RT} \right)
\]

\[
= \frac{\delta}{\delta \rho_R} \left( \eta \frac{\rho_T^k}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RO} - \frac{\delta}{\delta \rho_R} \frac{\eta v_{\text{ro}} v_{\text{ro}, \rho} \rho_T^k}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RO} - \frac{\delta}{\delta \rho_R} \frac{k\eta}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \rho_T^{k-1} \tilde{n}_{RT} \right)
\]

Where,

\[
\frac{\delta}{\delta \rho_R} \frac{\eta \rho_T^k}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RO} = \left( \frac{-k\eta \rho_T^{k-1}}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RT} - \frac{2\eta \rho_T^k}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \left( -\tilde{n}_{RO} + \frac{v_{\text{ro}}}{a_{\max} \rho_s} v_{\text{ro}, \rho} \tilde{n}_{RO} \right) \right) \tilde{n}_{RO}
\]

\[
= \frac{-k\eta \rho_T^{k-1}}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RT} \cdot \tilde{n}_{RO} + \frac{2\eta \rho_T^k}{(\rho_s - \rho_R - \rho_m(v_{\text{ro}}))^2} \tilde{n}_{RO} \cdot \tilde{n}_{RO}
\]

Combining (A.35), (A.37), (A.38) and (A.39) yields
\[
\frac{\delta U}{\delta \vec{\rho}_{DFR}} = \left( m - 1 \right) m \alpha_p \rho_T^{-2} \vec{n}_{RT} \cdot \vec{n}_{RT} + \frac{-k \eta \rho_T^{k-1}}{\left( \rho_s - \rho_R - \rho_m (v_{RO}) \right)} \vec{n}_{RT} \cdot \vec{n}_{RO} + \frac{2 \eta \rho_T^k}{\rho_s - \rho_R - \rho_m (v_{RO})} \vec{n}_{RO} \cdot \vec{n}_{RO}
\]

\[+ \left( \frac{k \eta \rho_T^{k-1}}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))} \right) \vec{n}_{RT} \cdot \vec{n}_{RO} + \left( \frac{2 \eta v_{RO} v_{RO} \rho_T^k}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))^2} \right) \vec{n}_{RO} \cdot \vec{n}_{RO} \]

\[+ \left( \frac{k \eta v_{RO} v_{RO} \rho_T^{k-1}}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))} \right) \vec{n}_{RT} \cdot \vec{n}_{RO} + \left( \frac{2 \eta v_{RO} v_{RO} \rho_T^k}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))^2} \right) \vec{n}_{RO} \cdot \vec{n}_{RO} \]

\[+ \left( \frac{k \eta v_{RO} v_{RO} \rho_T^{k-1}}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))} \right) \vec{n}_{RT} \cdot \vec{n}_{RO} + \left( \frac{2 \eta v_{RO} v_{RO} \rho_T^k}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))^2} \right) \vec{n}_{RO} \cdot \vec{n}_{RO} \]

\[= \left( m - 1 \right) m \alpha_p \rho_T^{-2} + \left( k - 1 \right) k \eta \left( \frac{1}{\rho_s - \rho_R - \rho_m (v_{RO})} - \frac{1}{\rho_T} \right) \rho_T^{k-2} \vec{n}_{RT} \cdot \vec{n}_{RT} \]

\[+ \left( \frac{-k \eta \rho_T^{k-1}}{\rho_s - \rho_R - \rho_m (v_{RO})} \right) \vec{n}_{RT} \cdot \vec{n}_{RO} + \vec{n}_{RO} \cdot \vec{n}_{RT} \]

\[+ \left( \frac{k \eta v_{RO} v_{RO} \rho_T^{k-1}}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))} \right) \vec{n}_{RT} \cdot \vec{n}_{RO} + \left( \frac{k \eta v_{RO} v_{RO} \rho_T^{k-1}}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))} \right) \vec{n}_{RO} \cdot \vec{n}_{RT} \]

\[+ \left( \frac{2 \eta \rho_T^k}{\rho_s - \rho_R - \rho_m (v_{RO})} \right) \vec{n}_{RT} \cdot \vec{n}_{RO} \]

\[+ \left( \frac{2 \eta v_{RO} v_{RO} \rho_T^k}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))^2} \right) \vec{n}_{RO} \cdot \vec{n}_{RO} \]

\[\text{(A.40)}\]

Differentiation: to position and velocity

\[\frac{\delta U}{\delta \vec{\rho}_{DFR}} = \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( U_{att} + U_{rep} \right) \quad \text{(A.41)}\]

\[\frac{\delta U_{att}}{\delta \vec{\rho}_{DFR}} = \frac{\delta U_{att}}{\delta \vec{\rho}_{DFR}} = \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( \alpha_p \rho_T^m + \alpha_v v_{RT}^m \right) \]

\[= \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( m \alpha_p \rho_T^{m-1} \vec{n}_{RT} \right) \]

\[= 0 \quad \text{(A.42)}\]

\[\frac{\delta U_{rep}}{\delta \vec{\rho}_{DFR}} = \frac{\delta U_{rep}}{\delta \vec{\rho}_{DFR}} = \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( \eta \left( \frac{1}{\rho_s - \rho_R - \rho_m (v_{RO})} - \frac{1}{\rho_T} \right) \rho_T^k \right) \]

\[= \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( \frac{\eta \rho_T^k}{\rho_s - \rho_R - \rho_m (v_{RO})^2} \vec{n}_{RO} \right) - \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( \frac{\eta v_{RO} v_{RO} \rho_T^k}{\rho_s a_{max} (\rho_s - \rho_R - \rho_m (v_{RO}))^2} \vec{n}_{RO} \right) - \frac{\delta}{\delta \vec{\rho}_{DFR}} \left( \frac{1}{\rho_s - \rho_R - \rho_m (v_{RO})} - \frac{1}{\rho_T} \right) \rho_T^{k-1} \vec{n}_{RT} \]

Where,
Combining (A.42), (A.44), (A.45) and (A.46) yields

\[
\frac{\delta U}{\delta \rho \vec{v}^R} = \frac{\delta U}{\delta \rho \vec{v}^R} = \frac{-k\eta \rho^R \rho^k}{\alpha_{\max}(\rho_s - \rho_R - \rho_m(v_{RO}))} \cdot \vec{n}_{RO} \cdot \vec{n}_{RT} + \left( \frac{2\eta \rho^R \rho^k}{\alpha_{\max}(\rho_s - \rho_R - \rho_m(v_{RO}))} \right) \vec{n}_{RO} \cdot \vec{n}_{RO} + \left( \frac{\eta \rho^R \rho^k}{\alpha_{\max}(\rho_s - \rho_R - \rho_m(v_{RO}))} \right) \vec{n}_{RO \perp} \cdot \vec{n}_{RO \perp} \quad (A.47)
\]

Differentiation: twice to velocity

\[
\frac{\delta U}{\delta \vec{v}^R} = \frac{\delta}{\delta \vec{v}^R} \left( U_{att} + U_{rep} \right) \quad (A.48)
\]

\[
\frac{\delta U_{att}}{\delta \vec{v}^R} = \frac{\delta}{\delta \vec{v}^R} \left( \frac{1}{\rho_0} \right) \rho^k \quad (A.49)
\]

\[
\frac{\delta U_{rep}}{\delta \vec{v}^R} = \frac{\delta}{\delta \vec{v}^R} \left( \frac{1}{\rho_0} \right) \rho^k \quad (A.50)
\]

Combining (A.49) and (A.50) yields

\[
\frac{\delta U_{rep}}{\delta \vec{v}^R} = \left( (n - 1) n \alpha_v v_{RT}^R - 2 \right) \vec{n}_{RT} \cdot \vec{n}_{VRT} + \left( \frac{\eta \rho^k}{\alpha_{\max}(\rho_s - \rho_R - \rho_m(v_{RO}))} \right) \vec{n}_{RO} \cdot \vec{n}_{RO} \quad (A.51)
\]
Appendix A5: Data structures and code tree

Data structures
Reality [Timestep, Time]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestep</td>
<td>length of iteration step</td>
</tr>
<tr>
<td>Time</td>
<td>Current time</td>
</tr>
</tbody>
</table>

Field [L B R U]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Left boundary</td>
</tr>
<tr>
<td>B</td>
<td>Bottom boundary</td>
</tr>
<tr>
<td>R</td>
<td>Right boundary</td>
</tr>
<tr>
<td>U</td>
<td>Upper boundary</td>
</tr>
</tbody>
</table>

Target [x y vx vy]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x position of target</td>
</tr>
<tr>
<td>y</td>
<td>y position of target</td>
</tr>
<tr>
<td>vx</td>
<td>velocity in x direction of target</td>
</tr>
<tr>
<td>vy</td>
<td>velocity in y direction of target</td>
</tr>
</tbody>
</table>

Robot [xR yR vxR vyR a_app D-Moved NX NY Maxvelocity MaxAcceleration Mass]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xR</td>
<td>x position of robot</td>
</tr>
<tr>
<td>yR</td>
<td>y position of robot</td>
</tr>
<tr>
<td>vxR</td>
<td>velocity in x direction of robot</td>
</tr>
<tr>
<td>vyR</td>
<td>velocity in y direction of robot</td>
</tr>
<tr>
<td>a_app</td>
<td>applied acceleration</td>
</tr>
<tr>
<td>D-Moved</td>
<td>distance travelled by robot</td>
</tr>
<tr>
<td>NX</td>
<td>x direction of Newton direction</td>
</tr>
<tr>
<td>NY</td>
<td>y direction of Newton direction</td>
</tr>
<tr>
<td>Maxvelocity</td>
<td>maximum velocity of robot</td>
</tr>
<tr>
<td>MaxAcceleration</td>
<td>maximum acceleration of robot</td>
</tr>
<tr>
<td>Mass</td>
<td>mass of robot</td>
</tr>
</tbody>
</table>

Constants [m n k alpha_p alpha_v etha rhoR rho0 amax]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>power of position part of attractive potential function</td>
</tr>
<tr>
<td>n</td>
<td>power of velocity part of attractive potential function</td>
</tr>
<tr>
<td>k</td>
<td>power of rhot in repulsive potential function</td>
</tr>
<tr>
<td>alpha_p</td>
<td>weigh factor of position part of attractive potential function</td>
</tr>
<tr>
<td>alpha_v</td>
<td>weigh factor of velocity part of attractive potential</td>
</tr>
<tr>
<td>etha</td>
<td>weigh factor of repulsive potential</td>
</tr>
<tr>
<td>rhoR</td>
<td>radius of robot modified with radia of obstacles</td>
</tr>
<tr>
<td>rho0</td>
<td>influence zone of obstacles</td>
</tr>
<tr>
<td>amax</td>
<td>maximum acceleration of robot</td>
</tr>
</tbody>
</table>

Obstacles [xO yO vxO vyO rhoR penaltyfactor]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xO</td>
<td>x position of obstacle</td>
</tr>
<tr>
<td>yO</td>
<td>y position of obstacle</td>
</tr>
<tr>
<td>vxO</td>
<td>x component of velocity of obstacle</td>
</tr>
<tr>
<td>vyO</td>
<td>y component of velocity of obstacle</td>
</tr>
<tr>
<td>rhoR</td>
<td>radius of robot modified with radius of obstacles</td>
</tr>
<tr>
<td>penaltyfactor</td>
<td>Factor if obstacle is rated as significant</td>
</tr>
</tbody>
</table>
**Code tree**

- Algorithm
- Start
  - FieldGhost
  - ConfigurationFunction
  - NormalFunction
  - Force
  - % Hessian Local %
  - Gradients
  - EscapeForce
  - Movement
  - Logger
  - ContourPlot
  - % GradientTracing %
- end

- OutputPlots
Appendix A.6 M-files

function [ ] = Algorithm( )
%ALGORITHM Obstacle Evasion Algorithm Mk. V
% Mouton
% T.Wilschut, Sept-Okt-2011
% Algorithm which evades stationary obstacles.

% Reset
clear all; close all; pause on;

% Input
% [Timestep, time]
Reality = [1/32 0] ;
% Field size [Left Bottom Right Upper]
Field = [0 0.0 5 5] ;
% Target configuration [x y vx vy]
Target = [4 4 0 0] ;
% Robot configuration
% [xR yR vxR vyR a_app D Moved NX NY Maxvelocity MaxAcceleration Mass]
Robot = [0.0 0.0 0.0 0.0 0 0 0 0 4 5 1] ;
% Constants [m n k alpha_p alpha_v etha rhoR rho0 amax]
Constants = [2 2 2 10 sqrt(20) 1 0.5 1 5] ;
% Mobile obstacle velocity
VO = 4 ;
% Obstacle [xO yO vxO vyO radius penaltyfactor]
Obstacles = [
    2.5 0.3 0 0 0.5 0  ;
    1.25 0 0.5 1  ;
    2.2 1.2 0 0.5 1  ;
    3.0 2.7 0 0.5 0  ;
    1.8 4.1 0 0.5 0  ;
]
% 'Hesian test'
...2.5 1.4 0 0 0.5 0  ;
...1.4 2.4 0 0 0.5 0  ;
...1.5 1.5 0 0 0.5 0  ;
% Local minimum
% One obstacle
...1.5 1.5 0 0 0.5 0  ;
% Two obstacles
...2.1 1.5 0 0 0.5 0  ;
...1.5 2.1 0 0 0.5 0  ;
% Three obstacles
...1.5 2.4 0 0 0.5 0  ;
...2.4 1.5 0 0 0.5 0  ;
...2.3 2.3 0 0 0.5 0  ;
% Movement Test
7.1 7 -sqrt(VO^2/2) -sqrt(VO^2/2) 0.5 0  ;
...2.5+VO 1.5 -VO 0 0.5 0  ;
...2.5+VO 2.5+VO -sqrt(VO^2/2) sqrt(VO^2/2) 0.5 0  ;
...3 1.5-VO 0 VO 0.5 0  ;
-0.5*VO-0.2 -0.5*VO sqrt(VO^2/2) sqrt(VO^2/2) 0.5 0 ; ...

% Random
...10 10 0 0 0.5 0 ; ...
...0.5 3.2 0 0 0.5 0 ; ...
...1.5 1.45 0 0 0.5 0 ; ...
...1.8 3.1 0 0 0.5 0 ; ...
...2.8 2.8 0 0 0.5 0 ; ...
...3 1 0 0 0.5 0 ; ...
...4.5 2.8 0 0 0.5 0 ; ...
];

% Number of obstacles in field
[n,~] = size(Obstacles);

%% Force vectors
% Position part of artificial force
Fp = zeros(2,1) ;
% Velocity part of artificial force
Fv = zeros(2,1) ;

%% Inverse Inertia matrix
M = inv(eye(2,2)) ;

%% External forces
N = zeros(2,1) ;

%% Ghost obstacle information [x y r p]
GhostObstacles = zeros(2,4) ;
GhostObstacles(:,[1,2]) = NaN(2,2) ;

%% Initialize Log
[ RobotLog, ObstaclesLog, GhostObstaclesLog, TargetLog ] = ... 
Logger( Reality, Robot, Obstacles, GhostObstacles,Target) ;

%% Start
% Error = pTar - pR
Error = inf ;
% Maximum number of time steps
MaxSteps = 150 ;
% T(0)
SimTime = 0 ;

%% Simulation Step
% Opening avifile
aviobj = avifile('Local minimum, one obstacle.avi','... 
'compression','None');

while Error > 0.1 && Reality(2) < MaxSteps
tic;

%% Produce Ghost Obstacles for Field Borders
[ GhostObstacles ] = ... 
FieldGhosts ( Robot, Reality, Field, GhostObstacles);

%% Determine all relative velocities and positions
[ Re_velocities, Difference_vectors ] = ... 
ConfigurationFunction( Robot, Target,Obstacles,GhostObstacles);

%% Determining all normalised vectors in the desired directions
[ normalvectors ] = ...
Normalfunction(Re_velocities, Difference_vectors);

% Determine Forces
[Forces, Att_Force, Rep_Force, Obstacles, Fp, Fv, Frep, Frep'] = ...
  Force(Constants, normalvectors, Difference_vectors, Re_velocities, ...
  Obstacles, Fp, Fv);

% Determine Local invhess
% [InvHess] = Hessian_Local( Constants, Difference_vectors, ...
% normalvectors, Re_velocities, Obstacles, GhostObstacles);

% Unity Hessian
InvHess = eye(2,2);

% Determine gradient of field
[Gradient, Modified_Frep] = Gradients(Forces, InvHess);

% Checking if local minimum is present and adding escape force if required
[Gradient, Es_Force] = EscapeForce(Constants, Re_velocities, ...
  normalvectors, Robot, Gradient, Att_Force, Rep_Force, ...
  Difference_vectors, Obstacles, Modified_Frep);
% Es_Force = [0 0]';

% Newton Direction
Robot([7, 8]) = Gradient'/norm(Gradient);

% Movement of robot
[Robot, Obstacles, Reality, Target] = Movement(Reality, ...
  Difference_vectors, Robot, M,N, Obstacles, Target);

% Determine Error
Error = norm(Difference_vectors(:, 1));

% Advance one step in time
SimTime = SimTime + toc;

% Output Logs
[RobotLog, ObstaclesLog, GhostObstaclesLog, TargetLog] = ...
  Logger(Reality, Robot, ...
  Obstacles, GhostObstacles, Target, ...
  RobotLog, ObstaclesLog, GhostObstaclesLog, TargetLog);

% Obstacle Proximity
Reality(2) = Reality(2)+1;
Proximity(1, Error = norm(Difference_vectors(:, ID+1));
for ID = 1:n
  Proximity(2, ID+1) = Error;
end

% Plot movement of robot
ContourPlot( Constants, Gradient, Att_Force, Modified_Frep', Es_Force, ...
  Robot, Obstacles, GhostObstacles, Target, Reality, Rep_Force, ...
  Frep, Frep');

% Plot of velocity of robot
% [Robot] = GradientTracing(Reality, Robot);

% Acquiring frame for .avi
F = getframe;
aviobj = addframe(aviobj, F);
end

% Closing avifile
aviobj = close(aviobj);

%% Output

OutputPlots(Reality, RobotLog, TargetLog, Obstacles,...
    Constants, Proximity, SimTime,n,ObstaclesLog);

%% Denitialize
pause off

%% End of simulation
end
function [ GhostObstacles ] = FieldGhosts( Robot, Reality, Field, GhostObstacles )
%FIELD Creates Ghosts of field edges to contain Robot.
% L Mouton
% IN: Robot Structure
% IN: Field Structure
% IN: GhostObstacles
% OUT: GhostObstacles (updated)

%%% Add Collision Ghost
if Reality(2) > 0
    GhostObstacles = WallGhost(Robot, Field, GhostObstacles);
end
end

%%% Ghost on the Wall
function [GhostObstacles] = WallGhost(Robot, Field, GhostObstacles)
PreviousLocation = GhostObstacles(1,[1,2]);
for n = 1:4
    %Find intercept point on border
    % Bottom and upper side
    if mod(n,2) == 0
        % Time to collision = ((Bottom or upper limit) - yR)/ vyR
        ttc = (Field(n)-Robot(2))/Robot(4);
        % x position of wallGhost = xR + (Time to collision)*vxR
        x = Robot(1)+ttc*Robot(3);
        % time to collision > 0 and x> left boundary and x < right boundary
        if ttc > 0 && x > Field(1) && x < Field(3)
            GhostObstacles(1,[1,2]) = [x,Field(n)]+...
                UnitDirection(Robot([3,4]))*0.1;
        end
    else
        % Left and right side
        ttc = (Field(n)-Robot(1))/Robot(3);
        % y position of wall ghost = yR + (Time to collison)*vyR
        y = Robot(2)+ttc*Robot(4);
        % time to collision > 0 and y > lower boundary and < upper boundary
        if ttc > 0 && y > Field(2) && y < Field(4)
            GhostObstacles(1,[1,2]) = [Field(n),y]+...
                UnitDirection(Robot([3,4]))*0.1;
        end
    end
end

%%% Size & Weight (if close enough)
% Influence area of field
if norm(GhostObstacles(1,[1,2])-Robot([1,2])) < 1
    % Radius
    GhostObstacles(1,3) = 0.1;
    % Weight
    GhostObstacles(1,4) = min(1, ...
        1/norm(Robot([1,2])-GhostObstacles(1,[1,2])));
else
    %Decay on previous location.
    GhostObstacles(1,[1,2]) = PreviousLocation;
    GhostObstacles(1,4) = max(GhostObstacles(1,4)-0.05,0);
end
end
function [ Re_velocities, Difference_vectors ] = ConfigurationFunction...
    ( Robot, Target, Obstacles,GhostObstacles)
% Configuration
% This function calculates all difference vectors and relative velocities
% T.Wilschut
% 04-10-2011
%-------------------------------------------------------------------------------------

% Calculating relative velocities
% Relative velocity between robot and target
[Re_velocities1] = AddRe_velocities(Robot, Target);

% Relative velocities between robot and obstacles
[Re_velocities2] = AddRe_velocities(Robot, Obstacles);

% Relative velocity between robot and GhostObstacles
[Re_velocities3] = AddRe_velocities(Robot, GhostObstacles);

% All relative velocities
Re_velocities = [Re_velocities1,Re_velocities2,Re_velocities3];

% Function to calculate relative velocities
function [Re_velocities] = AddRe_velocities(Robot, Add)
    [n,~] = size(Add);
    Re_velocities = zeros(2,n);
    for i = 1:n;
        Re_velocities(:,i) = [ (Add(i,3)-Robot(3)) ... 
                               (Add(i,4)-Robot(4)) ];
    end
end

% Calculating difference vectors
% Difference vector to target
[Difference_vector1] = AddDifference_vector(Robot, Target);

% Difference vectors to obstacles
[Difference_vector2] = AddDifference_vector(Robot, Obstacles);

% Difference vectors to GhostObstacles
[Difference_vector3] = AddDifference_vector(Robot, GhostObstacles);

% All difference vectors
Difference_vectors = [Difference_vector1,Difference_vector2,...
                        Difference_vector3];

% Function for calculating difference vectors
function [Difference_vector] = AddDifference_vector(Robot, Add)
    [n,~] = size(Add);
    Difference_vector = zeros(2,n);
    for i = 1:n;
        Difference_vector(:,i) = [ (Add(i,1)-Robot(1)) ... 
                                   (Add(i,2)-Robot(2)) ];
    end
end

end
function [ normalvectors ] = Normalfunction(Re_velocities,...
    Difference_vectors)

%% Normal Function
% This function calculates all difference vectors with unit length
% T. Wilschut
% Okt -2011
%---------------------------------------------------------------------

%% Scaling all difference vectors to unit length
[~,n] = size(Difference_vectors);
normalvector = zeros(2,n);
for i = 1:n;
    normalvector(:,i) = Difference_vectors(:,i)/ max(...
        norm(Difference_vectors(:,i)), 0.001);
end

%% Scaling Relative velocity vector between robot and target to unit length
nvrt      = zeros(2,1);
if norm(Re_velocities(:,1)) ~= 0;
    nvrt(:,1) = Re_velocities(:,1)/max(norm(Re_velocities(:,1)),0.001);
else
    nvrt(:,1) = normalvector(:,1);
end

%% Calculating perpendicular unit vectors
Perpendicular_vectors = zeros(2,n);
for i = 1:n;
    Perpendicular_vectors(:,i) = [(normalvector(2,i)) ... 
        -(normalvector(1,i))]
end

%% All unit vectors
normalvectors = [nvrt, normalvector, Perpendicular_vectors];
end
    constants, normalvectors, Difference_vectors, ...
    Re_velocities, Obstacles, Fp, Fv)

% Force
% Calculates all attractive and repulsive forces
% T. Wilschut
% Okt. 2011
%--------------------------------------------------------------------------

%% Constants = [m n k alpha_p alpha_v etha]
    m       = constants(1);
    n       = constants(2);
    k       = constants(3);
    alpha_p = constants(4);
    alpha_v = constants(5);
    etha    = constants(6);
    rhor    = constants(7);
    rho0    = constants(8);
    amax    = constants(9);

    [~,q] = size(Difference_vectors);
    [r,~] = size(Obstacles);

%% Calculation of the attractive force
    Fatt1 = m*alpha_p*norm(Difference_vectors(:,1))^(m-1)...
        *normalvectors(:,2);
    rhot = norm(Difference_vectors(:,1));
    Fatt2 = (n*alpha_v*norm(Re_velocities(:,1))^(n-1))*(normalvectors(:,1));

%% Calculation of repulsive forces
    Frep1 = zeros(2,q-1);
    Frep2 = zeros(2,q-1);
    Frep3 = zeros(2,q-1);
    Frep4 = zeros(2,q-1);
    for i = 1:q-1;
        if isnan(norm(Difference_vectors(:,i+1)))
            continue;
        else
            % Distance to obstacle
            rhos = norm(Difference_vectors(:,i+1));

            % Component of the relative velocity between robot and obstacle
            % pointing in the direction of nROi
            vro = (Re_velocities(:,i+1)'*normalvectors(:,i+2));

            % Velocity component perpendicular to vro
            vrop = Re_velocities(:,i+1)'*normalvectors(:,i+q+2);

            % Dynamic influence zone rhod
            rhom = vro^2/(2*amax);

        % Checking if relative velocity in the direction of the robot towards the obstacle between robot and obstacle
        % is negative and robot is within influence region
if (rhos-rhor-rhom)< rho0 && vro < 0

% Position gradient
Frep1(:,i) = ( (-etha*rhot^k)/(rhos-rhor-rhom) )\...  
*normalvectors(:,i+2);

Frep2(:,i) = ( (etha*vro*vrop*rhot^k)/max((rhos*amax*...  
(rhos-rhor-rhom)^2),10^-9) )*normalvectors(:,i+q+2);

Frep3(:,i) = ( k*etha*( (1/max((rhos-rhor-rhom),10^-9))...  
- 1/rho0 )*(rhot^(k-1)) )*normalvectors(:,2);

% Velocity gradient
Frep4(:,i) = ( (etha*vro*rhot^k)/(amax*...  
(rhos-rhor-rhom)^2) )*normalvectors(:,i+2);

% Penalty factor for controur plot
if i<=r;
    Obstacles(i,6) =1;
end

% Let repulsive force decay for smoother velocity profile
elseif (rhos) < rho0

% Position gradient
Frep1(:,i) = 0.20*(Fp-Fatt1');

Frep2(:,i) = [0 0]';
Frep3(:,i) = [0 0]';

% Velocity gradient
Frep4(:,i) = 0.20*(Fv-Fatt2');

else
    Frep1(:,i) = 0.0;
    Frep2(:,i) = 0.0;
    Frep3(:,i) = 0.0;
    Frep4(:,i) = 0.0;

% Penalty factor for contour plot
if i<=r;
    Obstacles(i,6) =0;
end
end
end
end

%% All Forces

% Position derivative of attractive potential function
Fattp = Fatt1';
% Position derivative of repulsive potential function
Frepp = [sum(Frep1(1,:)) sum(Frep1(2,:))]
    +[sum(Frep2(1,:)) sum(Frep2(2,:))]
    +[sum(Frep3(1,:)) sum(Frep3(2,:))];
% Velocity derivative of attractive potential function
Fattv = Fatt2';
% Velocity derivative of repulsive potential function
Frepv = [sum(Frep4(1,:)) sum(Frep4(2,:))];
% Total force due to position information
Fp = Fattp+Frepp;

% Total velocity force
Fv = Fattv+Frepv;

% Force matrix
Forces = [Fattp;Fattv;Frepp;Frepv];

% Attractive part of potential function differentiated to pR and vR
Att_Force = Fatt1+Fatt2 ;

% Repulsive part of potential function differentiated to pR and vR
Rep_Force = [sum(Frep1(1,:)) sum(Frep1(2,:))]'...  
+ [sum(Frep2(1,:)) sum(Frep2(2,:))]'...  
+ [sum(Frep3(1,:)) sum(Frep3(2,:))]'...  
+ [sum(Frep4(1,:)) sum(Frep4(2,:))]' ;

end
function [ InvHess ] = ...
    Hessian_Local( Constants, Difference_vectors,normal_vectors,...
        Re_velocities, Obstacles, GhostObstacles)

function Hess = AddHess(x,Constants,Difference_vectors,normal_vectors,...
    Re_velocities, Hess, Obstacles, GhostObstacles)

%% HESSIAN Local
% Calculates the local inverse Hessian function
%
% T. Wilschut
% Okt - 2011
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Hessian
    Hess = zeros(2,2);

%% Hessian of repulsive potential
% Add Real Obstacles
    x=0;
    Hess = AddHess(x,Constants,Difference_vectors,normal_vectors,...
        Re_velocities, Hess, Obstacles, GhostObstacles);
% Add Ghost Obstacles
    x=1;
    Hess = AddHess(x,Constants,Difference_vectors,normal_vectors,...
        Re_velocities, Hess, Obstacles, GhostObstacles);

%% Correct & Invert Hessian
    a = Hess(1,1);
    b = Hess(1,2);
    c = Hess(2,1);
    d = Hess(2,2);

%Check & Correct
    if det(Hess) <= 0
        mu = ( - (a + d) + sqrt( (a+d)^2 - 4 * (a*d-b*c) ) ) /2;
        Hess = [a+1.01*mu, b; c, d+1.01*mu];
    end

%Invert Hessian
    a = Hess(1,1);
    b = Hess(1,2);
    c = Hess(2,1);
    d = Hess(2,2);

% Safe guard against NaN components in InvHess
    if Hess(1,1) == 0
        InvHess = eye(2,2);
    else
        InvHess = (1 / (a*d - b*c) * [ d -b ; -c a ]); end
end

%% Add Hessian of something to function.
function [Hess] = AddHess(x,constants,Difference_vectors,normal_vectors,...
    Re_velocities, Hess, Add1,Add2)

% Defining constants
    k       = constants(3);
    etha    = constants(6);
    rhor    = constants(7);
    rho0    = constants(8);
    amax    = constants(9);

% Choosing the right difference vectors
if x ==0;
    [q,~]  = size(Add1);
    [r,~]  = size(Add2);
    s = q+r+2;
    y =2;
    z =q+1;
else x ==1;
    [q,~]  = size(Add1);
    [r,~]  = size(Add2);
    s = q+r+2;
    y =q+2;
    z =q+r+1;
end

for i = y:z;
    if isnan(normal_vectors(:,i+1))
        continue;
    else

        % Defining function variables
        % Distance to target
        rhot = norm(Difference_vectors(:,1));
        % Distance to obstacle i
        rhos = norm(Difference_vectors(:,i));
        % Unitvector towards target
        nrt = normal_vectors(:,2);
        % Unitvector pointing to obstacle i
        nro = normal_vectors(:,i+1);
        % Relative velocity component in direction of nro
        vro = ((Re_velocities(:,i))'*nro);
        % Distance needed to reduce relative velocity
        rhom = vro^2/(2*amax);
        % Unitvector perpendicular to nro
        nrop = normal_vectors(:,i+s);
        % Velocity component in direction of nrop
        vrop = ((Re_velocities(:,i))'*nrop);

        % Hessian components
        if (rhos-rhor-rhom) < rho0 && vro < 0;
            %----------------------------------------------------------
            a = ( (k-1)*k*etha*(1/(rhos-rhor-rhom) -...\       
                1/rho0)*rhot^(k-2) )*nrt'*nrt...\       
                + ( (-k*etha*rhot^(k-1))/(rhos-rhor-rhom)^2 )...\       
                *( nrt'*nro + (nro)'*nrt )...\       
                + ( ( k*etha*vro*vrop*rhot^(k-1) )/(rhos^2*amax...\       
                    *(rhos-rhor-rhom)^2 )*(nrt)'*nrop...\       
                + ( (-k*etha*vro*vrop*rhot^(k-1))/(rhos*amax...\       
                    *(rhos-rhor-rhom)^2 )*(nrop)'*nrt...\       
                + ( (2*etha*rhot^k)/(rhos-rhor-rhom)^3 )...\       
                    *(nro)'*nro...\       
                + ( (2*etha*vro^n2*vrop^2*rhot^k)/(rhos*amax^2*...\       
                        (rhos-rhor-rhom)^3 )*(nrop)'*nrop;\       
            %----------------------------------------------------------

            b = ( (-k*etha*vro*rhot^(k-1))/...\       
                (amax*(rhos-rhor-rhom)^3 )*(nro)'*nrt...\       
                + ( (2*etha*rhot^k*vro)/...\       
                    (amax*(rhos-rhor-rhom)^3 )*(nro)'*nro...\       
                %----------------------------------------------------------

    end

end
\[ + \left( \frac{\text{etha} \cdot \text{vro} \cdot \text{rhot}^k}{\text{rhos} \cdot \text{amax} \cdot (\text{rhos} - \text{rhor} - \text{rhom})^2} \right) \cdot (\text{nrop})' \cdot \text{nrop}; \]

% New Hessian

Hess = Hess + [a b; b d];

end

end

end
function [ Gradient, Modified_Frep ] = Gradients( Forces, InvHess)

%% GRADIENTS
% Calculates the desired direction of motion
% T.Wilschut
% Oct- 2011
% -------------------------------------------------------
% Multiplying repulsive Forces with inverse Hessian
Modified_Force = InvHess*Forces([3 4],:);

% Modified repulsive force
Modified_Frep = [sum(Modified_Force(:,1)) sum(Modified_Force(:,2))];

% Calculating gradient
Gradient = (Forces(1,:)+Forces(2,:) + Modified_Frep )';...

end
function [Gradient,Es_force] = EscapeForce(Constants,Re_velocities,...
    normalvectors,Robot, Gradient ,Att_Force, Rep_Force, ...
    Difference_vectors , Obstacles, Modified_Frep)

% This function introduces an escape force if a local minimum is present
% Tim Wilschut
% Okt-2011

%% Conditions for local minimum not the target position
A = norm(Gradient)/norm(Rep_Force) < 0.3;

% Angle between Fatt and Gradient
Att_angle = (acos ((Att_Force'*Gradient)/
    (norm(Att_Force)*norm(Gradient))));

% Angle between Frep and Gradient
Rep_angle = (acos((Rep_Force'*Gradient)/
    (norm(Rep_Force)*norm(Gradient))));

B = cos(Att_angle - Rep_angle) < -cos(5/6*pi);

C = norm(Difference_vectors(:,1)) > 0.5;

local_minimum = A & B & C;

%% Choosing obstacle which is closest
rhos = inf;
for i = 1:length(Obstacles(:,1));
    if rhos> norm(Difference_vectors(:,i+1));
        rhos = (norm(Difference_vectors(:,i+1)));
        q = i;
    end
end

%% Radius of robot
rhor = Constants(7);

%% Relative velocity between robot and obstacle
vro = Re_velocities(:,q)''*normalvectors(:,q+2);

%% Distance needed to reduce relative velocity to zero
rhom = vro^2/2*Robot(10);

%% Velocity of robot
VR = Robot([3,4])'';

%% Difference vector from target to obstacle q
D_OT = Difference_vectors(:,q+1) - Difference_vectors(:,1);

%% Local minimum present and robot is not near target position
if local_minimum && norm(Difference_vectors(:,1)) > norm(D_OT)...;
    nFrep = UnitDirection(Rep_Force);
    npFrep = [(nFrep(2,1)) , -(nFrep(1,1))]' ;

    % Checking angle between npFrep and VR
    if acos((VR''*npFrep)/(norm(VR)*norm(npFrep))) > 0.5*pi;

end
npFrep = -npFrep;
end

% Checking if escape force does not steer robot over field boundary
if Robot(1)< 0.3;
    npFrep(1,1) = abs(npFrep(1,1));
elseif Robot(1) > 4.7;
    npFrep(1,1) = -abs(npFrep(1,1));
elseif Robot(2) < 0.3;
    npFrep(2,1) = abs(npFrep(2,1));
elseif Robot(2) > 4.7;
    npFrep(2,1) = -abs(npFrep(2,1));
end

% Order of attractive force
z = log(norm(Rep_Force))/log(10)';

% Calculating escape force
Es_force = 100^z*(1/(rhos-rhor-rhom)^2)*npFrep;

% Calculating new Gradient
Gradient = Gradient + Es_force;
else
    Es_force = [0 0]';
end
    Gradient = Gradient;
end
function [ Robot, Obstacles, Reality, Target ] = Movement(Reality, Difference_vectors, Robot, M, N, Obstacles, Target);

% Movement
% This function determines the required acceleration to accurately trace
% the desired direction of motion
%
% T. Wilschut
% Okt.2011

% Maximum velocity
Vmax = Robot(9);
% Maximum acceleration
amax = Robot(10);
% Maximum Available control force
Tau0 = Robot(11)*amax;
% Newton direction
Nd = Robot([7, 8])';
% Robot velocity
VR = Robot([3, 4])';
% Distance to target
rhot = norm(Difference_vectors(:, 1));
% Desired velocity
Vdes = min(Vmax)*Nd; % sqrt(2*amax*rhot)
% Velocity error
dV = -VR + Vdes;
% Control force to be applied
tau = Tau0*(dV./max(norm(dV), 0.000001));
% Resulting acceleration
if norm(tau) == 0;
a_app = [0 0]';
else
a_app = (M)*(tau - N);
end

% Resulting new position
Robot([1, 2]) = Robot([1, 2]) + (VR*Reality(1) + 0.5*a_app*Reality(1)^2)';

% Storing total distance travelled
Robot(6) = Robot(6) + norm(VR*Reality(1) + 0.5*a_app*Reality(1)^2);

% Resulting new velocity
Robot([3, 4]) = Robot([3, 4]) + (a_app*Reality(1))';
if norm(Robot([3, 4])) >= Vmax;
    Robot([3, 4]) = Vmax*UnitDirection(Robot([3, 4]));
    a_app = [0 0]';
end

% Total acceleration
if norm(a_app) == 0;
    Robot(5) = 0;
else
    Robot(5) = (a_app)'*abs(UnitDirection(a_app));
end

% Obstacles
% Move
Obstacles(:, [1, 2]) = Obstacles(:, [1, 2]) + Obstacles(:, [3, 4])*Reality(1);

% Target
Target([1, 2]) = Target([1, 2]) + Target([3, 4])*Reality(1);
end
function [ RobotLog, ObstaclesLog, ...  
    GhostObstaclesLog, TargetLog ] = ...
Logger( Reality, Robot, Obstacles, GhostObstacles, Target, ...
    RobotLog, ObstaclesLog, GhostObstaclesLog, TargetLog )

%% Logger
%% Simply saves Arrays in an Array with an additional time dimension.
%%
%% Mouton
%% T. Wilschut Okt-2011
%-----------------------------------------------------------------------------------

%% Log State

%% First step safety.
if Reality(2) == 0
    Reality(2) = 1;
end

%% Storing data in log
RobotLog(Reality(2),:) = Robot(1,:);
ObstaclesLog(Reality(2),:,:) = Obstacles(:,:);
GhostObstaclesLog(Reality(2),:,:) = zeros(2,4);
TargetLog(Reality(2),:) = Target(1,:);
end
function [] = ContourPlot( Constants, Gradient, Att_Force, Modified_Frep, ...
    Es_Force, Robot, Obstacles, GhostObstacles, Target, Reality, Rep_Force, ...
    Frepp, Freppv)

% Contour plot
% Plots movement of robot
% Snap shots of robot motion can be taken
% Plot force vectors
%
% !! Note that plotted potential field is still Old potential function of
% Mouton and not the new potential field with velocity information !!!
%
% L Mouton
% T. Wilschut Okt-2011
%
% %-------------------------------------------------------------------------

%% Build Grid of Target Locations
Granularity = 10;
FieldX = [1:5*Granularity];
FieldY = [1:5*Granularity];
ContourPoints = zeros(max(FieldX), max(FieldY));

for i = 1:1:max(FieldX) % X
    for j = 1:1:max(FieldY) % Y

        % Target Function
        FunctionValue = norm([i/Granularity, j/Granularity] - ...
            [Target(1), Target(2)])^2;

        % Add Obstacles
        FunctionValue = ...
            AddObs(FunctionValue, Obstacles, 5, 6, Granularity, i, j);

        % Add GhostObstacles
        FunctionValue = ...
            AddObs(FunctionValue, GhostObstacles, 3, 4, Granularity, i, j);

        % Set Value
        ContourPoints(i,j) = FunctionValue;
    end
end

%% Defining vectors
% Desired direction of motion
Ftot = (UnitDirection(Gradient)'+[Robot(1) Robot(2)])*Granularity;
% Attractive force vector
Fatt = (UnitDirection(Att_Force)'+[Robot(1) Robot(2)])*Granularity;
% Repulsive force vector
Frep1 = (UnitDirection(Rep_Force)'+[Robot(1) Robot(2)])*Granularity;
% Modified repulsive vector
Frep2 = (UnitDirection(Modified_Frep)'+[Robot(1) Robot(2)])*Granularity;
% Repulsive position vector
Frepp = (UnitDirection(Frepp)+[Robot(1) Robot(2)])*Granularity;
% Repulsive velocity vector
Freppv = (UnitDirection(Freppv)+[Robot(1) Robot(2)])*Granularity;
% Escape force vector
Fes = (UnitDirection(Es_Force)'+[Robot(1) Robot(2)])*Granularity;
% Position of robot
Pr = Robot([1,2])*Granularity;

%% Plot field with potential field isolines
figure(1);
contour(FieldX, FieldY, ContourPoints', [0:2.5:100]);
%Nice-ify picture.
axis([0 max(FieldX) 0 max(FieldY)]);
axis square;
xlabel('x [dm]')
ylabel('y [dm]')

Additional info
% Plot robot position
plot(Robot(1)*Granularity, Robot(2)*Granularity, 'bo', 'MarkerSize', 5);
plot(Robot(1)*Granularity, Robot(2)*Granularity, 'b*', 'MarkerSize', 5);

% Plot target position
plot(Target(1)*Granularity, Target(2)*Granularity, 'go', 'MarkerSize', 5);

% Plot attractive force vector
draw_arrow(Pr,Fatt,0.3,[0 1 0]);

% Plot repulsive force vector
draw_arrow(Pr,Frep1,0.3,[1 0 0]);

% Plot position part of repulsive force
draw_arrow(Pr,Frepp,0.3,[0 1 1]);

% Plot velocity part of repulsive force
draw_arrow(Pr,Frepv,0.3,[1 0 1]);

% Plot modified repulsive force vector
draw_arrow(Pr,Frep2,0.3,[1 1 0]);

% Plot escape force vector
draw_arrow(Pr,Fes,0.3,[0 0 0]);

% Plot desired direction of motion vector
draw_arrow(Pr,Ftot,0.3,[0 0 1]);

%Show Obstacles
[n,~] = size(Obstacles);

% Generate circles around obstacle
% Circle with radius rhor
x1 = [-Constants(7):0.01:Constants(7)];
y1 = zeros(size(x1));
for i = 1:length(x1);
y1(i) = sqrt(Constants(7)^2-x1(i)^2);
end

% Circle with radius rho0
x2 = [-Constants(8):0.01:Constants(8)];
y2 = zeros(size(x2));
for i = 1:length(x2);
y2(i) = sqrt(Constants(8)^2-x2(i)^2);
end

for i = 1:n
plot((x1+Obstacles(i,1))*Granularity,(y1+Obstacles(i,2))*Granularity, '-r', 'MarkerSize', 2);
plot((x1+Obstacles(i,1))*Granularity,(-y1+Obstacles(i,2))*Granularity, '-r', 'MarkerSize', 2);
plot((x2+Obstacles(i,1))*Granularity,(y2+Obstacles(i,2))*Granularity, '-r', 'MarkerSize', 2);
plot((x2+Obstacles(i,1))*Granularity,(-y2+Obstacles(i,2))*Granularity, '-r', 'MarkerSize', 2);
end
hold off
% 3D-SurfPlot
% Flatten for Surf Plot
figure(1);
for i = 1:1:max(FieldX) %X
    for j = 1:1:max(FieldY) %Y
        if ContourPoints(i,j) > 200
            ContourPoints(i,j) = 200;
        end
    end
end
figure(1);
surf(FieldX,FieldY,ContourPoints');

% Additional info
axis([0 max(FieldX) 0 max(FieldY)]);
axis square;
xlabel('x [dm]')
ylabel('y [dm]')

hold on
plot(Robot(1)*Granularity, Robot(2)*Granularity,'bo','MarkerSize',5);
plot(Robot(1)*Granularity, Robot(2)*Granularity,'b*','MarkerSize',5);
plot(Reality(3)*Granularity, Reality(4)*Granularity,'go','MarkerSize',5);
hold off

%% Save Snapshots (Optional)
if Reality(2) >= 30 && Reality(2) <= 35;
    if mod(Reality(2),1) == 0;
        Fname = ['StaObsMin_' int2str(Reality(2)) '.tif']
        set(gcf, 'paperunits', 'centimeters', 'paperposition', [0 0 15 15])
        print('-dtiff','-r300',Fname)
    end
end

pause(0.0); %No pause = skip to end.
end

%% Add Something to contourplot
function [FunctionValue] = ...
    AddObs(FunctionValue, Add, r, pf, Granularity, i, j)
    [n,~] = size(Add);
    for q = 1:n
        if ~isnan(Add(q,1))
            % Boundary Function
            Boundary = -norm([Add(q,1),Add(q,2)])-...
                [i/Granularity, j/Granularity])^2 + Add(q,r)^2;
            % Read Penalty Factor of Obstacle
            PenaltyFactor = Add(q,pf);
            % Add to Function
            if Boundary >= 0 && Add(q,pf) ~= 0;
                % Within obstacle, FunctionValue = infinity
                FunctionValue = inf;
            else
                % Outside obstacle, add Barrier Function
                FunctionValue = FunctionValue + PenaltyFactor/(-Boundary);
            end
        end
    end
end

%% Arrow Plot
function [] = draw_arrow(startpoint, endpoint, headsize, color)

% accepts two [x y] coords and one double headsize

v1 = headsize*(startpoint-endpoint)/2.5;

theta = 22.5*pi/180;
theta1 = -1*22.5*pi/180;
rotMatrix = [cos(theta) -sin(theta); sin(theta) cos(theta)];
rotMatrix1 = [cos(theta1) -sin(theta1); sin(theta1) cos(theta1)];

v2 = v1*rotMatrix;
v3 = v1*rotMatrix1;
x1 = endpoint;
x2 = x1 + v2;
x3 = x1 + v3;

fill([x1(1) x2(1) x3(1)], [x1(2) x2(2) x3(2)], color);
plot([startpoint(1) endpoint(1)], [startpoint(2) endpoint(2)],...
     'linewidth', 1, 'color', color);
end
function [] = OutputPlots(Reality, RobotLog, TargetLog, Obstacles,...
    Constants, Proximity, SimTime,n,ObstaclesLog);

% Output plots
% Robot,Target and Obstacle paths
% Distance from robot to target and obstacles
% Velocity plot
% Acceleration plot
%
% T. Wilschut
% Okt.2011
%----------------------------------------------------------------------------------

%% Output Plots
figure();
hold on
%Target Location
plot(TargetLog(:,1),TargetLog(:,2),'black','MarkerSize',10);
%Robot Location
plot(RobotLog(:,1),RobotLog(:,2),'b-','LineWidth',2);

% Generate circle around obstacle
x1 = [-Constants(7):0.01:Constants(7)];
y1 = zeros(size(x1));
for i = 1:length(x1);
    y1(i) = sqrt(Constants(7)^2-x1(i)^2);
end
x2 = [-Constants(8):0.01:Constants(8)];
y2 = zeros(size(x2));
for i = 1:length(x2);
    y2(i) = sqrt(Constants(8)^2-x2(i)^2);
end

%Obstacle Locations
for i = 1:n
    % Stationary obstacle
    plot((x1+Obstacles(i,1)),(y1+Obstacles(i,2)),'-r','LineWidth',2);
    plot((x1+Obstacles(i,1)),(-y1+Obstacles(i,2)),'-r','LineWidth',2);
    plot((x2+Obstacles(i,1)),(y2+Obstacles(i,2)),'-r','LineWidth',2);
    plot((x2+Obstacles(i,1)),(-y2+Obstacles(i,2)),'-r','LineWidth',2);

    % Dynamic obstacle
    plot((ObstaclesLog(:,1)),(ObstaclesLog(:,2)+Constants(7)),'-r','LineWidth',2);
    plot((ObstaclesLog(:,1)),(ObstaclesLog(:,2)-Constants(7)),'-r','LineWidth',2);
    plot((ObstaclesLog(:,1)),(ObstaclesLog(:,2)+Constants(8)),'-r','LineWidth',2);
    plot((ObstaclesLog(:,1)),(ObstaclesLog(:,2)-Constants(8)),'-r','LineWidth',2);

    axis([0 5 0 5]); axis square;
    xlabel('x [m]','Fontsize',24); ylabel('y [m]','Fontsize',24);
    title('Robot path','Fontsize',24)
end
legend('Target','Robot path','Obstacle','rho0');
grid
hold off
%% Velocity Plot
figure
hold on
plot([1:Reality(2)-1],sqrt(RobotLog(:,3).^2+RobotLog(:,4).^2),'LineWidth',2);
plot([1:Reality(2)-1],RobotLog(:,9),'r--','LineWidth',2);
plot(0,4.5)
ylabel('v [m/s]', 'FontSize', 24); xlabel('t [steps]', 'FontSize', 24);
title('Velocity of robot during navigation', 'FontSize', 24);
legend('vR', 'vmax')
grid
hold off

%% Acceleration Plot
figure
hold on
plot([1:Reality(2)-1],RobotLog(:,5),'b-', 'LineWidth',2);
plot([1:Reality(2)-1],RobotLog(:,10),'r--', 'LineWidth',2);
plot([1:Reality(2)-1],-RobotLog(:,10),'r--', 'LineWidth',2);
plot(0,5.5)
ylabel('a [m/s^2]', 'FontSize', 24); xlabel('t [steps]', 'FontSize', 24);
title('Acceleration of robot during navigation', 'FontSize', 24);
legend('aR', 'amax')
grid
hold off

%% Proximity Plot
figure
hold on
for ID = 1:n+1
    if ID == 1
        plot([1:Reality(2)],Proximity(:,ID),'g-','LineWidth',2);
    else
        plot([1:Reality(2)],Proximity(:,ID),'r-','LineWidth',2);
    end
end
plot([1,Reality(2)],[Constants(7),Constants(7)],'b-','LineWidth',2);
plot([1,Reality(2)],[Constants(8),Constants(8)],'b--','LineWidth',2);
ylabel('d [m]', 'FontSize',24); xlabel('t [steps]', 'FontSize',24);
title('Distance from robot to target and obstacles', 'FontSize',24);
grid
hold off

%% Output
MinimalDistance = min(min(Proximity(:,[2:n+1])))
AverageVelocity = sum(sqrt(RobotLog(:,3).^2+RobotLog(:,4).^2))/Reality(2)
TotalTime = Reality(2)*Reality(1)
AverageStepTime = SimTime/Reality(2)
DistanceMoved = RobotLog(Reality(2)-1,6)
end