Design and implementation of a
generic intercept strategy
for soccer robots

J.H. Baten
S088816

CST 2011.066

Traineeship report

Coach(es): R. Hoogendijk
Supervisor: M.J.G. van de Molengraft

Eindhoven University of Technology
Department of Mechanical Engineering
Systems and Control

Eindhoven, September, 2011
The Robocup MSL (Middle Size League) is a robot soccer league. In the Robocup MSL a team consists of 5 robots playing soccer autonomously. A very important aspect during a soccer match is intercepting the ball. Intercepting the ball occurs in many different situations. For example when the opponent is in ball possession or when the ball moves free over the field. The traineeship assignment is to design and implement a more successful strategy that is situation dependent, easy to extend and easy to change. The existing software of a soccer robot is created in Matlab Simulink with C code. An important subject during the traineeship is to do research on how and with which program to implement the intercept strategy. The first step in the design of the intercept strategy, is analyzing the many different situations that can occur during a MSL soccer match. With the situations defined the intercept strategy is designed in flowchart form. There are many different programs to implement the intercept strategy in flowchart form, in code. After comparing the pros and cons of these programs, the intercept strategy in flowchart form is implemented in ROS (Robot Operating System).
Contents

1 Introduction

2 Analysis of the situations
   2.1 Defining situations

3 Intercept transitions
   3.1 Transitions related to the turtle
      3.1.1 Is it possible for the turtle to intercept the ball?
      3.1.2 Is there a teammate behind the turtle?
      3.1.3 Is the turtle between the own goal and opponent?
      3.1.4 Is the opponent close to the turtle?
   3.2 Transitions related to the opponent
      3.2.1 Is the opponent in ball possession?
      3.2.2 Is it possible for the opponent to intercept the ball?
      3.2.3 On which side is the opponent with ball possession?
      3.2.4 On which half is the opponent with ball possession?
   3.3 Transitions related to the ball
      3.3.1 Who has the best chance to intercept the ball?
      3.3.2 What is the velocity of the ball?
      3.3.3 What is the position of the ball?
      3.3.4 Is the turtle in the trajectory of the ball?
      3.3.5 Who has last touched the ball?
      3.3.6 Is the ball moving towards the turtle?

4 Intercept strategy in flowchart form

5 Implementation choice
   5.1 Requirements on the software
   5.2 Programs to implement the intercept strategy in code
      5.2.1 Dia2code
      5.2.2 ALTOVA
      5.2.3 BOUML
      5.2.4 Visual Paradigm SDE for Eclipse
      5.2.5 ROS
      5.2.6 Pros and cons of the programs
   5.3 Motivation for choosing ROS
Chapter 1

Introduction

The Robocup MSL (Middle Size League) is a robot soccer league. In the Robocup MSL a team consists of 5 robots playing soccer autonomously. Tech United is a team of enthusiastic students, PhD students and other members that participates in the MSL. The homebase of Tech United is the TU/e (Technische Universiteit Eindhoven). During a MSL soccer match many different situations occur. The situations can be divided into three groups namely when the turtle is in ball possession, when the opponent is in ball possession and when neither the opponent or turtle is in ball possession. The current strategy is simply, drive towards ball and intercept regardless of the situation. During the MSL soccer matches it has been observed that this is not a successful intercept strategy. The traineeship assignment is to design and implement a more successful strategy that is situation dependent, easy to extend and easy to change.

The first step in the design of an intercept strategy is to define the requirements for the intercept strategy. The intercept strategy must satisfy the following requirements:

1. The intercept strategy should be easy to extend and easy to change
2. Intercept the ball in a strategic way
   (a) When there is no other teammember behind the turtle, defend with a strategy that prevents a scorings possibility for the opponent
   (b) When there is a teammember behind the turtle, defend with a strategy with the aim to intercept the ball
   (c) Defend away from goal
   (d) Defend with a different strategy when the ball is last touched by the opponent
   (e) Defend with a different strategy when it is not possible to intercept the ball
The second step is to implement the intercept strategy using a software development process. Software development processes are structures to help software engineers develop software. There are many methods for these processes, each describing approaches to a variety of tasks. During the traineeship, software is developed using the spiral method. The spiral method is a combination of a linear and iterative framework, and is used for large and complex projects. With this method the software is divided into smaller segments, providing more ease to changes during the development process. Each segment is developed with the same sequence of tasks, namely a requirement part, analysis part, design part and implementation part. The sequence of tasks is shown in Figure 1.1.

![Figure 1.1: Spiral model](image)

In the first sequence, the requirements are defined in as much detail as possible, the preliminary design is created and implemented. The second sequence is evolved using four steps, namely:

- The evaluation of the implemented preliminary design
- Define the requirements for the second design based on the weaknesses and fails of the preliminary design
- Implementation of the second design
- Testing the second design

The third sequence has the same steps as the second sequence. The number of sequences is software dependent.

The intercept strategy can be designed using many different approaches. One way is to design the intercept strategy in state-machine form, with the intercepts as the states. This is a good approach for simple strategies, but loses the overview due to the many transitions between the states when the strategy becomes more complex. For that reason this approach will not be used, because the intercept strategy must satisfy the requirement that the intercept strategy is easy to change and extend. Another method is to design the intercept strategy in flowchart form. The advantage of this method is that flowcharts are easy readable, easy to change and easy to extend.

In Chapter 2 the situations that can occur during a MSL soccer match are defined. In this chapter an example of a situation that can occur during the MSL soccer match is given in the form of a flowchart. In Chapter 3 all the transitions between the states are explained in detail with equations and sketches of the situations. Now that all the transitions are explained in detail, the intercept strategy is designed in flowchart form in Chapter 4. The software to implement the intercept strategy is chosen in Chapter 5. In this chapter, multiple programs are explained in detail and the pros and cons are gathered. At the end of Chapter 5 a motivation for the use of ROS is given. Chapter 6 provides an introduction to ROS, and an explanation on how the intercept strategy is implemented in ROS. In Chapter 7 the conclusion and recommendations are given.
Chapter 2

Analysis of the situations

2.1 Defining situations

A first step in the design of an intercept strategy is to define all the situations that can occur during a MSL soccer match. This requires looking at videos of the MSL soccer matches, and collecting information from the Tech United robocup team. As there are infinite many situations possible, clustering of situations in a number of classes is necessary. The clustering of situations is performed by distinguishing them according to the following properties:

1. Positions on the field of
   (a) Peer players
   (b) Opponents
   (c) Ball
2. Velocity of the ball

For each of these properties a limited number of possibilities is defined. For that reason the soccer field is divided into multiple areas, as shown in Figure 2.1. Furthermore the ball velocity is divided into three categories; low, medium and high. Low is defined as the velocity of the ball where the ball is almost stationary. Medium is defined as the velocity of the ball where it is possible for the turtle to intercept without bouncing back from the turtle. And high is defined as the velocity of the ball where it is not possible for the turtle to intercept without bouncing back from the turtle.
Analysis of the situations

The possibilities of the opponents position, turtle position, ball position and ball velocity are gathered in Table 2.1, 2.2, 2.3 and 2.4. With a combination of these tables, any situation that can occur in the field during a MSL football match can be described. An example is given in Figure 2.2(a), 2.2(b), 2.2(c) and 2.2(d). These 4 flowcharts together describe the situation shown in Figure 2.3.

**Table 2.1: possibilities: position opponent**

<table>
<thead>
<tr>
<th>wrt turtle</th>
<th>close</th>
<th>not close</th>
</tr>
</thead>
<tbody>
<tr>
<td>on which half</td>
<td>turtle</td>
<td>opponent</td>
</tr>
<tr>
<td>in which area</td>
<td>touch</td>
<td>goal</td>
</tr>
<tr>
<td>wrt ball</td>
<td>close</td>
<td>far</td>
</tr>
</tbody>
</table>

**Table 2.2: possibilities: position turtle**

<table>
<thead>
<tr>
<th>teammember behind turtle</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>on which half</td>
<td>turtle</td>
<td>opponent</td>
</tr>
<tr>
<td>in which area</td>
<td>touch</td>
<td>goal</td>
</tr>
<tr>
<td>wrt ball</td>
<td>close</td>
<td>far</td>
</tr>
</tbody>
</table>

**Table 2.3: possibilities: position ball**

<table>
<thead>
<tr>
<th>ball possession opponent</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>on which half</td>
<td>turtle</td>
<td>opponent</td>
</tr>
<tr>
<td>in which area</td>
<td>touch</td>
<td>goal</td>
</tr>
<tr>
<td>last touched by</td>
<td>turtle</td>
<td>opponent</td>
</tr>
</tbody>
</table>

**Table 2.4: possibilities: velocity ball**

<table>
<thead>
<tr>
<th>velocity ball</th>
<th>low</th>
<th>medium</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrt turtle</td>
<td>towards</td>
<td>away</td>
<td></td>
</tr>
<tr>
<td>wrt goal line</td>
<td>towards</td>
<td>away</td>
<td></td>
</tr>
<tr>
<td>wrt touch line</td>
<td>towards</td>
<td>away</td>
<td></td>
</tr>
</tbody>
</table>
2.1 Defining situations

Figure 2.2: Example of a situation
Figure 2.3: Example of a possible situation during a Robocup match
Intercept transitions

To make distinction between the defined situation classes, transitions need to be satisfied. The position vector and velocity vector of the opponent, turtle and ball are used in the equations to check whether a transition is true or false. The functions get_distance and get_angle that are available in the current software, are sometimes used to obtain a distance or angle between two point in the field. All the transitions are explained in detail below.

3.1 Transitions related to the turtle

3.1.1 Is it possible for the turtle to intercept the ball?

- Turtle cannot intercept ball:
  The turtle cannot intercept the ball, when the ball crosses the touch or goal line, before a turtle can intercept the ball. A graphical representation of the ball crossing the touch or goal line is shown in Figure 3.1.
To include the direction in which the ball is moving into Equation 3.1 till 3.8, \( \varphi \) is divided into four segments. As depicted in Figure 2.1 the distances from the center of the field to the touch line and goal line are 6.0m and 9.0m. These distances are used in the calculation of the distance between the ball and touch line and the distance between the ball and goal line. The distance between the ball and touch line and the distance between the ball and goal line, are

if \((0 < \varphi \leq 90)\) :

\[
x_{\text{out}} = -6.0 - x_{\text{ball}},
\]

\[
y_{\text{out}} = 9.0 - y_{\text{ball}}.
\]

elsif \((90 < \varphi \leq 180)\) :

\[
x_{\text{out}} = -6.0 - x_{\text{ball}},
\]

\[
y_{\text{out}} = -9.0 - y_{\text{ball}}.
\]

elsif \((180 < \varphi \leq 270)\) :

\[
x_{\text{out}} = 6.0 - x_{\text{ball}},
\]

\[
y_{\text{out}} = -9.0 - y_{\text{ball}}.
\]

else:

\[
x_{\text{out}} = 6.0 - x_{\text{ball}},
\]

\[
y_{\text{out}} = 9.0 - y_{\text{ball}}.
\]
3.1 Transitions related to the turtle

The time it takes the ball to cross the touch and goal line, is

\[ t_{x_{\text{out}}} = \text{abs}(x_{\text{out}}/\dot{x}_{\text{ball}}), \]  
\[ t_{y_{\text{out}}} = \text{abs}(y_{\text{out}}/\dot{y}_{\text{ball}}). \]  

(3.9) (3.10)

The time it takes the ball to go out, is

\[ t_{\text{ball}_{\text{out}}} = \text{the smallest value of } t_{x_{\text{out}}} \text{ and } t_{y_{\text{out}}}. \]  

(3.11)

The position where the ball is going out, is

\[ \text{ball}_{x_{\text{out}}} = x_{\text{ball}} + t_{\text{ball}_{\text{out}}} \cdot \dot{x}_{\text{ball}}, \]  
\[ \text{ball}_{y_{\text{out}}} = y_{\text{ball}} + t_{\text{ball}_{\text{out}}} \cdot \dot{y}_{\text{ball}}. \]  

(3.12) (3.13)

To include the direction in which the turtle is moving in the equations, the equations are divided into two parts. Namely, one part when the turtle already moves towards the position where the ball is going out, and one part when the turtle moves away from the position where the ball is going out. With the functions get_distance and get_angle, \( b \) and \( \varphi \) are obtained. A graphical representation of the turtle moving towards the position where the ball is going out, is shown in Figure 3.2. The time it takes for the turtle to position itself at the position where the ball is going out, is

![Figure 3.2: Turtle moving towards the position where the ball is going out.](image_url)
- **Turtle already moves towards the position where the ball is going out:**

The velocity of the turtle in the direction of the position where the ball is going out, is

$$v_{\text{turtle\_out}} = \cos \varphi * \vec{v}_{\text{turtle}}$$

$$= \cos \varphi * \sqrt{x_{\text{turtle}}^2 + y_{\text{turtle}}^2}.$$ (3.14)

The time it takes to accelerate the turtle from the initial turtle velocity to the maximum turtle velocity, is

$$t_{\text{turtle\_accel}} = (v_{\text{turtle\_max}} - v_{\text{turtle\_out}})/a_{\text{turtle\_max}}.$$ (3.15)

The distance covered by the turtle during the turtle acceleration time, is

$$d_{\text{turtle\_accel}} = 0.5 * a_{\text{turtle\_max}} * t_{\text{turtle\_accel}}^2 + v_{\text{turtle\_out}} * t_{\text{turtle\_accel}}.$$ (3.16)

The time it takes the turtle to overcome the remaining distance between the turtle and the position where the ball is going out, is

$$t_a = (b - d_{\text{turtle\_accel}})/v_{\text{turtle\_max}}.$$ (3.17)

The total time it takes for the turtle to cover the distance between the turtle and the position where the ball is going out, is

$$t_{\text{turtle}} = t_{\text{turtle\_accel}} + t_a.$$ (3.18)
3.1 Transitions related to the turtle

- Turtle moves away from the position where the ball is going out:
The velocity of the turtle in the direction of the position where the ball is going out, is

\[ v_{\text{turtle\_out}} = \cos \varphi \times \vec{v}_{\text{turtle}} = \cos \varphi \times \sqrt{\dot{x}_{\text{turtle}}^2 + \dot{y}_{\text{turtle}}^2}. \]  \hspace{1cm} (3.19)

Because the turtle is moving away from the position where the ball is going out, the turtle first needs to decelerate before it can move towards the position where the ball is going out. The time it takes to decelerate the turtle from the initial turtle velocity to the minimum turtle velocity, is

\[ t_{\text{turtle\_decel}} = \frac{v_{\text{turtle\_out}}}{a_{\text{turtle\_max}}}. \]  \hspace{1cm} (3.20)

Now that the turtle has decreased its velocity to the minimum turtle velocity, the turtle can increase its velocity towards the position where the ball is going out to the maximum turtle velocity. The time it takes to accelerate the turtle from the minimum turtle velocity to the maximum turtle velocity, is

\[ t_{\text{turtle\_accel}} = \frac{v_{\text{turtle\_max}}}{a_{\text{turtle\_max}}}. \]  \hspace{1cm} (3.21)

The distance covered by the turtle during the turtle deceleration time, is

\[ d_{\text{turtle\_decel}} = 0.5 \times a_{\text{turtle\_max}} \times t_{\text{turtle\_decel}}^2. \]  \hspace{1cm} (3.22)

The distance covered by the turtle during the turtle acceleration time, is

\[ d_{\text{turtle\_accel}} = 0.5 \times a_{\text{turtle\_max}} \times t_{\text{turtle\_accel}}^2. \]  \hspace{1cm} (3.23)

The time it takes the turtle to overcome the remaining distance between the turtle and the position where the ball is going out, is

\[ t_a = \frac{(b - d_{\text{turtle\_accel}} + d_{\text{turtle\_decel}}) / v_{\text{turtle\_max}}.}{} \]  \hspace{1cm} (3.24)

The total time it takes for the turtle to cover the distance between the turtle and the position where the ball is going out, is

\[ t_{\text{turtle}} = t_{\text{turtle\_accel}} + t_a + t_{\text{turtle\_decel}}. \]  \hspace{1cm} (3.25)

The turtle cannot intercept the ball when \( t_{\text{turtle}} \) is larger than \( t_{\text{ball\_out}} \)

- Turtle can intercept the ball:
The turtle can intercept the ball when \( t_{\text{turtle}} \) is smaller than \( t_{\text{ball\_out}} \)
3.1.2 Is there a teammember behind the turtle?

- Teammember behind turtle:
  There is a teammember behind the turtle, when the distance between the turtle and the own goal is larger, than the distance between the other turtles and the own goal. With the functions get_distance and get_angle $b_1$, $b_2$, $\varphi_1$, and $\varphi_2$ are obtained. A graphical representation of turtle 1 having a teammember behind him, is shown below in Figure 3.3.

![Figure 3.3: Teammember behind turtle](image)

To include the direction in which the turtle is moving in the equations, the equations are divided into two part. One part is when the turtle already moves towards the goal. The other part is when the turtle moves away from the goal. The equations to determine if the transition is true or false, are the same as Equation 3.14 till 3.25 used in the transition “Turtle cannot intercept the ball”, except that the position where the ball is going out is replaced by the center of the own goal.

- No teammember behind turtle:
  There is no teammember behind the turtle, when the distance between the turtle and the own goal is smaller, than the distance between the other turtles and the own goal. The equations to determine if the transition is true or false, are the same as Equation 3.14 till 3.25 in the transition “Turtle cannot intercept the ball”, except that the position where the ball is going out is replaced by the center of the own goal.
### 3.1.3 Is the turtle between the own goal and opponent?

- The position of the turtle is between the goal and the opponent with ball possession:
  The turtle is between the own goal and the opponent with ball possession, when the distance between turtle and the own goal is smaller than the distance between the opponent with ball possession and the Tech United goal, and \( \alpha \) is smaller than 10 degrees, where \( \alpha = |\varphi_1 - \varphi_2| \).
  With the functions \text{get}_\text{distance} \text{ and } \text{get}_\text{angle} \ b_1, b_2, \varphi_1 \text{ and } \varphi_2 \text{ are obtained. A graphical representation of the case that the turtle is between the goal and the opponent with ball possession, is shown in Figure 3.4.}

- Turtle’s position is not between the goal and the opponent with ball possession:
  The turtle is not between the own goal and the opponent with ball possession, when the distance between turtle and the own goal is equal or larger than the distance between the opponent with ball possession and the Tech United goal or \( \alpha \) is larger then 10 degrees.
3.1.4 Is the opponent close to the turtle?

- Opponent close to the turtle:
  The opponent is close to the turtle, when the opponent is within the predefined radius $r$ from the turtle. With the function `get_distance`, $b$ is obtained. A graphical representation of the opponent close to the turtle, is shown in Figure 3.5.

- Opponent not close to the turtle:
  The opponent is not close to the turtle, when the opponent is outside the predefined radius $r$ from the turtle.

Figure 3.5: Opponent is close to the turtle
3.2 Transitions related to the opponent

3.2.1 Is the opponent in ball possession?

- Ball possession opponent:
The opponent has possession of the ball, when

\[
\begin{align*}
  x_{\text{opp}} - x_{\text{ball}} &< \epsilon \\
  y_{\text{opp}} - y_{\text{ball}} &< \epsilon \\
  \dot{x}_{\text{opp}} - \dot{x}_{\text{ball}} &< \epsilon \\
  \dot{y}_{\text{opp}} - \dot{y}_{\text{ball}} &< \epsilon
\end{align*}
\]

(3.26) (3.27) (3.28) (3.29)

A graphical representation of the opponent in ball possession, is shown in Figure 3.6.

- No ball possession opponent:
The opponent is not in possession of the ball when one of the Equation 3.26 till 3.29 is not satisfied.
3.2.2 Is it possible for the opponent to intercept the ball?

- Opponent cannot intercept the ball:
The opponent cannot intercept the ball, when the ball crosses the touch or goal line, before the opponent can intercept the ball. The equations are the same as the equations used in the transition "Turtle cannot intercept the ball".

- Opponent can intercept the ball:
The opponent can intercept the ball, when it is possible for the opponent to intercept the ball, before it crosses the touch or goal line. The equations are the same as the equations used in the transition "Turtle can intercept the ball".

3.2.3 On which side is the opponent with ball possession?

- The opponent with ball possession is on the left side of the field:
The opponent with ball possession is on the left side of the field when the $x$ coordinate of the opponent is negative.

- The opponent with ball possession is on the right side of the field:
The opponent with ball possession is on the right side of the field when the $x$ coordinate of the opponent is positive.

3.2.4 On which half is the opponent with ball possession?

- The opponent with ball possession is on opponents half of the field:
The opponent with ball possession is on opponents half of the field when the $y$ coordinate of the opponent is positive.

- The opponent with ball possession is on the Tech United half of the field:
The opponent with ball possession is on turtles half of the field when the $y$ coordinate of the opponent is negative.
3.3 Transitions related to the ball

3.3.1 Who has the best chance to intercept the ball?

- Opponent has a good chance to intercept the ball before turtle:
  With the functions get_distance and get_angle \( b_1, b_2, \varphi_1, \varphi_2, \varphi_3 \) and \( \varphi_4 \) are obtained.

The velocity of the turtle in the direction of the ball position, is

\[
v_{\text{turtle, ball}} = \cos \varphi_1 \times v_{\text{turtle}}
\]  

(3.30)

The velocity of the ball in the direction of the turtle position, is

\[
v_{\text{ball, turtle}} = \cos \varphi_2 \times v_{\text{ball}}
\]  

(3.31)

The velocity of the opponent in the direction of the ball position, is

\[
v_{\text{opp, ball}} = \cos \varphi_3 \times v_{\text{opp}}
\]  

(3.32)

The velocity of the ball in the direction of the opponent position, is

\[
v_{\text{ball, opp}} = \cos \varphi_4 \times v_{\text{ball}}
\]  

(3.33)

To include the direction in which the turtle, opponent and ball are moving in the equations, the equations are divided into 4 parts, these 4 parts are then each divided into 2 parts.

- **Turtle already moves towards the position of the ball:**
  A graphical representation of the case that the turtle and opponent are already moving towards the ball and the ball moves towards them, is shown in Figure 3.7.

![Figure 3.7: Turtle and opponent moving towards ball and ball is moving towards the opponent and the turtle](image-url)
The time it takes to accelerate the turtle from the initial turtle velocity to the maximum turtle velocity, is

\[ t_{\text{turtle, accel}} = (v_{\text{turtle, max}} - v_{\text{turtle, ball}})/a_{\text{turtle, max}}. \]  

(3.34)

The distance covered by the turtle during the turtle acceleration time, is

\[ d_{\text{turtle, accel}} = 0.5 \cdot a_{\text{turtle, max}} \cdot t^2_{\text{turtle, accel}} + v_{\text{turtle, ball}} \cdot t_{\text{turtle, accel}}. \]  

(3.35)

The distance covered by the ball during the turtle acceleration time, is

\[ d_{\text{ball, turtle}} = v_{\text{ball, turtle}} \cdot t_{\text{turtle, accel}}. \]  

(3.36)

\* When the ball moves towards the position of the turtle
The total time it takes the turtle to intercept the ball, is

\[ t_{\text{turtle}} = (b_1 - d_{\text{ball, turtle}} - d_{\text{turtle, accel}})/(v_{\text{turtle, max}} + v_{\text{ball, turtle}}) + t_{\text{turtle, accel}}. \]  

(3.37)

\* When the ball moves away from the position of the turtle
The total time it takes the turtle to intercept the ball, is

\[ t_{\text{turtle}} = (b_1 + d_{\text{ball, turtle}} - d_{\text{turtle, accel}})/(v_{\text{turtle, max}} - v_{\text{ball, turtle}}) + t_{\text{turtle, accel}}. \]  

(3.38)

\* Turtle moves away from the position of the ball:
Because the turtle is moving away from the position of the ball, the turtle first needs to decelerate before it can move towards the position of the ball. The time it takes to decelerate the turtle from the initial turtle velocity to the minimum turtle velocity, is

\[ t_{\text{turtle, decel}} = v_{\text{turtle, out}}/a_{\text{turtle, max}}. \]  

(3.39)

Now that the turtle has decreased its velocity to the minimum turtle velocity, the turtle can increase its velocity towards the position of the ball to the maximum turtle velocity. The time it takes to accelerate the turtle from the minimum turtle velocity to the maximum turtle velocity, is

\[ t_{\text{turtle, accel}} = v_{\text{turtle, max}}/a_{\text{turtle, max}}. \]  

(3.40)

The distance covered by the turtle during the turtle deceleration time, is

\[ d_{\text{turtle, decel}} = 0.5 \cdot a_{\text{turtle, max}} \cdot t^2_{\text{turtle, decel}}. \]  

(3.41)

The distance covered by the turtle during the turtle acceleration time, is

\[ d_{\text{turtle, accel}} = 0.5 \cdot a_{\text{turtle, max}} \cdot t^2_{\text{turtle, accel}}. \]  

(3.42)

The distance covered by the ball during the turtle deceleration and acceleration time, is

\[ d_{\text{ball, turtle}} = v_{\text{ball, turtle}} \cdot (t_{\text{turtle, accel}} + t_{\text{turtle, decel}}). \]  

(3.43)

\* When the ball moves towards the turtle
The total time it takes the turtle to intercept the ball, is

\[ t_{\text{turtle}} = (b_1 - d_{\text{ball, turtle}} + d_{\text{turtle, decel}} - d_{\text{turtle, accel}})/(v_{\text{turtle, max}} + v_{\text{ball, turtle}}) + t_{\text{turtle, accel}} + t_{\text{turtle, decel}}. \]  

(3.44)
3.3 Transitions related to the ball

* When the ball moves in the opposite direction
  The total time it takes the turtle to intercept the ball, is
  \[ t_{turtle} = \left( b_1 + \Delta_{ball-turtle} + \Delta_{turtle-decel} - \Delta_{turtle-accel} \right) / \left( v_{turtle-max} - v_{ball-turtle} \right) + t_{turtle-accel} + t_{turtle-decel}. \] (3.45)

  **When the opponent already moves towards the ball:**
  The equations are the same as Equation 3.34 till 3.38 in the part “In the case that the turtle already moves towards the position of the ball”.

  **When the opponent moves in the opposite direction:**
  The equations are the same as Equation 3.39 till 3.45 in the part “In the case that the turtle moves in the opposite direction”.

Opponent has a good chance to intercept the ball before the turtle can intercept the ball, when \( t_{turtle} \) is larger then \( t_{opponent} \).

Turtle has a good chance to intercept the ball before opponent: Turtle has a good chance to intercept the ball before the opponent can intercept the ball, when \( t_{turtle} \) is smaller then \( t_{opponent} \).

3.3.2 What is the velocity of the ball?

* Velocity ball low:
  The velocity off the ball is low, when the velocity off the ball is lower than a preliminary defined velocity “low”.

* Velocity ball medium:
  The velocity off the ball is medium, when the velocity off the ball is lower than a preliminary defined velocity "high", and larger than the preliminary defined velocity "low".

* Velocity ball high:
  The velocity off the ball is high, when the velocity off the ball is higher than the preliminary defined velocity "high".

3.3.3 What is the position of the ball?

* Position ball in touch line area:
  The ball is in the touch line area, when the position of the ball is in the preliminary defined "touch line area”. The touch line area is the area close to the touch line, as can be seen in Figure 2.1

* Position ball in goal line area:
  The ball is in the goal line area, when the position of the ball is in the preliminary defined "goal line area”. The goal line area is the area close to the goal line, as can be seen in Figure 2.1

* Position ball not in touch line area or goal line area:
  The ball is not in the touch line area or goal line area, when the position of the ball is not in the preliminary defined “touch line area” or in the preliminary defined “goal line area”.

3.3.4 Is the turtle in the trajectory of the ball?

- Turtle is in the trajectory of the ball:
  Turtle is in the trajectory of the ball when the angle’s \( \varphi_1 \) and \( \varphi_2 \) are equal. With the functions get_distance and get_angle \( \varphi_1 \) and \( \varphi_2 \) are obtained. A graphical representation of the case that the turtle is in the trajectory of the ball, is shown below in Figure 3.8

\[
y + x + x - \varphi_1 = \varphi_2
\]

-v_{\text{ball}}
touch line
goal line

Figure 3.8: Position turtle in the trajectory of the ball

- Turtle is not in the trajectory of the ball:
  Turtle is in the trajectory of the ball when the angle’s \( \varphi_1 \) and \( \varphi_2 \) are unequal.

3.3.5 Who has last touched the ball?

- Ball is last touched by turtle:
  The ball is last touched by a turtle, when the memory function is true.

- Ball is last touched by opponent:
  The ball is last touched by the opponent, when the memory function is false.
3.3.6 Is the ball moving towards the turtle?

- Ball is moving towards turtle:
  The ball is moving towards the turtle, when $\varphi < 90$. With the functions get_angle $\varphi$ is obtained.
  A graphical representation of the ball moving towards the turtle, is shown in Figure 3.9.

- Ball is moving away from turtle:
  The ball is moving away from the turtle, when $\varphi \geq 90$. 

Figure 3.9: Ball is moving towards turtle
Chapter 4

Intercept strategy in flowchart form

A flowchart is a diagram that represents an algorithm or process. Each step in the algorithm or process is represented by a box and contains a short description of the process step. The boxes are connected with arrows showing the flow direction of the algorithm or process. Flowcharts are easy readable, easy to change and easy to extend. The intercept strategy is designed in flowchart form. The flowchart leads to one of the 15 intercepts. The 15 intercepts in the flowcharts and the description of these intercepts are gathered in Table 4.1. To reach an intercept in the intercept strategy flowchart, beginning from the initial state, transitions based on Chapter 3 need to be satisfied. The flowchart of the intercept strategy is shown in Figure 4.1. To furthermore increase the flowchart readability, the intercept strategy flowchart is divided into 8 subflowcharts. The 8 subflowcharts are shown in Appendix A. The transitions are chosen in this particular sequence to keep the flowcharts as small as possible. At the end of every subflowchart, the intercepts are defined. When the intercept is carried out, the flowchart starts again at its initial state.
### Table 4.1: Overview of all the intercepts used in the intercept strategy

<table>
<thead>
<tr>
<th>Intercepts</th>
<th>Intercept description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept 1</td>
<td>Position turtle next to the opponent on the right side</td>
</tr>
<tr>
<td>Intercept 2</td>
<td>Drive towards opponent and intercept</td>
</tr>
<tr>
<td>Intercept 3</td>
<td>Position turtle next to the opponent on the left side</td>
</tr>
<tr>
<td>Intercept 4</td>
<td>Make violation, drive hard against opponent</td>
</tr>
<tr>
<td>Intercept 5</td>
<td>Position the turtle between the opponent with ball possession and own goal</td>
</tr>
<tr>
<td>Intercept 6</td>
<td>Drive towards opponent with medium velocity</td>
</tr>
<tr>
<td>Intercept 7</td>
<td>Drive towards opponent with low velocity</td>
</tr>
<tr>
<td>Intercept 8</td>
<td>Drive backwards to reduce relative velocity between opponent and turtle</td>
</tr>
<tr>
<td>Intercept 9</td>
<td>Do nothing</td>
</tr>
<tr>
<td>Intercept 10</td>
<td>Drive towards the ball and intercept</td>
</tr>
<tr>
<td>Intercept 11</td>
<td>Position the turtle between the opponent and the ball</td>
</tr>
<tr>
<td>Intercept 12</td>
<td>Position turtle outside the trajectory of the ball</td>
</tr>
<tr>
<td>Intercept 13</td>
<td>Intercept the ball perpendicular to the touch line</td>
</tr>
<tr>
<td>Intercept 14</td>
<td>Intercept the ball perpendicular to the goal line</td>
</tr>
<tr>
<td>Intercept 15</td>
<td>Bounce the ball over the touch line</td>
</tr>
</tbody>
</table>

![Figure 4.1: Intercept strategy flowchart](image-url)
Chapter 5

Implementation choice

5.1 Requirements on the software

There are many different programs to implement the intercept strategy in code. To investigate which program is the best program to implement the intercept strategy in code, requirements need to be defined. The program must satisfy the following requirements:

- The program should be user friendly
- With the program it should be easy to extend or change the intercept strategy
- The program should be able to draw flowcharts or state-machines
- The program should be able to convert flowcharts to code
- The program should have a GUI (Graphical User Interface)

Some interesting programs are explained in more detail below.

5.2 Programs to implement the intercept strategy in code

5.2.1 Dia2code

Dia is a program that is used to draw UML’s, flowcharts and many other diagrams. Dia has special objects to help draw these diagrams. Dia2code is a small program that converts Dia diagrams to code. Dia2code reads the Dia file that contains a UML diagram and converts this to the code of choice. The Dia UML diagram can be converted to Ada, C, C++, Java, PHP, PHP5, Python, Ruby, shapefile, SQL and C#. Dia2code does not support reverse engineering from source code to UML diagrams. It also does not support round trip engineering from source code from/to UML class diagram. Dia2Code is constant in development and has a GNU GPL (General Public License).
5.2.2 ALTOVA

ALTOVA Umodel is a program that is used to draw UML diagrams or state-machine diagrams and converts these to code. The UML diagrams or state machine diagrams can be converted to Java, C#, or Visual Basic .NET code, that are compatible with eclipse or other popular java environments. ALTOVA Umodel supports reverse engineering from Java, C#, and Visual Basic .NET code to UML diagrams. With ALTOVA Umodel it is also possible to do Round trip engineering. The Altova UModel round-trip engineering capability reads the modified code and automatically updates UML diagrams, to keep the software synchronized. ALTOVA has no GNU GPL, so it is not for free.

5.2.3 BOUML

BOUML is a program that is used to draw UML diagrams and converts these to code. The UML diagrams can be converted to C++, Java, Idl, Php and Python. BOUML can be extended with plug-ins. The plug-ins that create code from UML diagrams and UML diagrams from code are already included in BOUML. BOUML is a free program and has a GNU GPL license.

5.2.4 Visual Paradigm SDE for Eclipse

SDE (Smart Development Environment) for eclipse is a program that is used to draw UML diagrams and converts these to code. The UML diagrams can be converted to 15 different programming languages. With SDE for eclipse reverse engineering is possible. Instant reverse engineering can convert 12 different programming languages into UML diagrams. SDE for eclipse can also generates state machine code from UML diagrams to Java, C#, VB.NET and C++. SDE for eclipse also supports round trip engineering. SDE for eclipse has no GNU GPL, so it is not for free.

5.2.5 ROS

ROS (Robot Operating System) is a software framework for robot software development. ROS provides libraries and tools to help software developers create robot applications. ROS is not a program that converts UML’s, flowcharts and other diagrams into code. With ROS, state-machines can be created with python code. ROS is open source, and thus it is possible to create and design libraries, packages and stacks. Questions can be asked on the ROS forum, and are almost immediately answered. ROS currently only works on UNIX-based platforms. There are many packages and stacks in ROS, each designed with a different functionality. For instance there are packages for GUI’s, state-machines, simulators, etc.
5.3 Motivation for choosing ROS

The motivation for choosing ROS to implement the intercept strategy, is that with ROS the intercept strategy is very easy to implement in state-machine form or flowchart form. With ROS it is also easy to make changes or extend the intercept strategy. Other advantages of ROS, are the ROS forum where questions can be asked and the possibility to start a GUI with the "smach_viewer" package. ROS is especially used for creating robot applications, and is open source.

### Table 5.1: Pros and cons of the programs

<table>
<thead>
<tr>
<th></th>
<th>Dia2code</th>
<th>AITOVA</th>
<th>BOUML</th>
<th>SDE for eclipse</th>
<th>ROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>User friendly</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+++</td>
</tr>
<tr>
<td>Easy to draw flowcharts</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Easy to extend or change flowcharts</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>Code generation</td>
<td>+</td>
<td>+ +</td>
<td>+ +</td>
<td>+ +</td>
<td>+ +</td>
</tr>
<tr>
<td>GUI</td>
<td>− − −</td>
<td>− − −</td>
<td>− − −</td>
<td>− − −</td>
<td>+ +</td>
</tr>
</tbody>
</table>

5.2.6 Pros and cons of the programs

To obtain a clear overview, the pros and cons of the programs mentioned above are gathered in Table 5.1.
Implementation of the intercept strategy in ROS

The intercept strategy is written in Python and implemented in ROS. ROS is a software framework for robot software development. ROS provides libraries and tools to help software developers create robot applications. Due to the tight schedule, the intercept strategy is only partly implemented in ROS. Although only a part of the intercept strategy is implemented, the problems that occur during implementation are the same as for implementing the whole intercept strategy and it demonstrates how it works in practice.

6.1 Introduction to ROS

ROS runtime "Graph" is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous communication over Services and asynchronous streaming of data over Topics. The basic Computation Graph concepts of ROS are nodes, master, messages, topics, and services, all of which provide data to the Graph in different ways. These concepts are explained in more detail:

- **Master**: The ROS Master provides name registration to the Computation Graph. Without the master communication between nodes is impossible.
- **Nodes**: Nodes are the processes that perform computations. A robot system contains many nodes. For example one node for the state-machine, one for the computations on which a transition in the state-machine is performed, one node for the graphical representation of the state-machine, etc.
- **Message**: A message describes what kind of data structure is send or received by nodes. For example, array, string, boolean, int, etc.
- **Topic**: A node sends out a message by publishing it to a given topic. A node that is interested in the data will subscribe to the appropriate topic. The topic is a name that is used to identify the content of the message.
- **Service**: Sometimes a publish/subscribe model as described above is not useful when an event based model is needed. Services are event based and provide the request/reply model. Services don’t need topics to communicate.
In Figure 6.1(a) a graphical representation a ROS publish/subscribe model is shown. Node 0 is publishing data to a topic named "chatter". When node 1 is interested in the data, it will subscribe to the topic "chatter". In Figure 6.1(b) a graphical representation of a ROS request/reply model is shown. In this figure node 0 is a client node, requesting a service from node 1, the service node. The service needs two message structures namely one for the request and one for the reply.

![Diagram of ROS communication models](image)

Figure 6.1: ROS communication models

### 6.2 Packages

Stacks and packages are used to keep software in ROS structured. A package may contain multiple nodes, topics, messages, launch files or anything else that is usefully organized together. A stack is a collection of packages that provide aggregate functionality. When installing one of the four default configurations, the most commonly used stacks are installed. Additionally, the stacks "executive_smach" and "executive_smach visualization" need to be installed for implementing the intercept strategy in state-machine form.

The stack "executive_smach" contains the package smach, which stands for State Machine. Smach is a python library to build state-machines. The stack "executive_smach visualization" contains the package Smach_viewer. Smach_viewer is a GUI that shows the states and transitions between states of the state-machine.
6.3 Implementation of the intercept strategy

A first step to implement the intercept strategy in ROS, is to create a new stack. The new stack contains three packages, one package for the nodes, one package for the topic message type and one package for the service message type. The three packages together give a fully functioning intercept strategy. The nodes contain the code to create the intercept strategy in state machine form. To keep the code readable, it is distributed over several nodes. There is a node that simulates the existing software and publishes data to a topic, a node "the client node" that subscribes data from the topic and contains the code that creates the states and connects all the states into a state-machine, and the remaining nodes are service nodes that perform services for the client node. The service nodes reply to the client node, if a transition between two states is met or not met. The code is written in python, because the imported package "smach", to create the state machine, is a python library. In Figure 6.2 a graphical representation of the nodes and communication between the nodes is shown.

With the "smach_viewer" package imported in the client node, a GUI can be started. The GUI shows the states and transitions of the intercept strategy flowchart. It also shows which subflowchart and state is active. In Figure 6.3 the GUI is shown. Given a situation on the field, an intercept is chosen by following the transitions and states in the flowchart. In Figure 6.3 the flowchart starts with the state "Ball possession opp?". In this example subflowchart 2 is active because, the opponent is in ball possession and there is no teammember behind the turtle. In subflowchart 2 the state "intercept 5" is active because it is possible for the turtle to intercept the ball and the turtle is not between the goal and the opponent. In the client node also a launch file is written. With a launch file all the nodes can be launch at the same time, instead of launching every node singly.
Implementation of the intercept strategy in ROS

Figure 6.3: GUI smach viewer
Chapter 7

Conclusion and Recommendations

7.1 Conclusion

During MSL soccer matches it has been observed that the current intercept strategy is not successful. The traineeship assignment is to design and implement an intercept strategy that is more successful, easy to change and easy to extend.

The intercept strategy is made in flowchart form instead of state-machine form, because flowcharts are easy readable and changes can be made more easily in flowcharts than in state-machines. The transitions between the flowcharts are checked if they are true or false with geometric formula’s. The intercepts at the end of every flowchart are situation dependent.

To implement the intercept strategy flowchart, research has been conducted on programs that could implement the intercept strategy in a generic way. After comparing the pros en cons, ROS was found to be the best program to implement the intercept strategy in flowchart form.

ROS is user friendly with enough documentation and tutorials to understand and learn the basic techniques and terminology used in ROS. Due to the tight schedule, the intercept strategy is only partly implemented in ROS. By running the launch file, communication between the node which simulates the existing software and the node which contains the intercept strategy, is established. The launch file also starts up a GUI. The GUI shows the intercept strategy flowchart, with all its states and transitions and lightens up the state that is active. The intercept strategy flowchart was implemented in ROS with the help of the ROS smach package. Smach is a python library that was very helpful in building the intercept strategy flowchart in ROS. Changes and extensions on the intercept strategy flowcharts are easy to make and straightforward.
7.2 Recommendations

The intercept strategy is partly implemented in ROS. To get a fully functioning intercept strategy it should be implemented completely in ROS. Before an implementation of the intercept strategy can be realized in the Turtles, further research should be performed on the communication between ROS nodes and the existing software. A solution could be to implement the existing code in a ROS node, or to design an interface which performs the communication between the ROS nodes and the existing software. If a solution is obtained for the communication problem it should strongly reduce and simplify new software developments due to the existence of stacks and packages in ROS that supports many different functionalities. There are for instance stacks and packages for the kinect camera. The intercept strategy should also be tested by simulations before implementing it on the Turtles. The simulations should test that the intercept strategy leads to an intercept for all the possible situations that could occur during a Robocup match. And the simulations should test that a situation leading to a intercept is the best possible intercept for that situation.
Flow-charts

Figure A.1: Subflowchart 1
Figure A.2: Subflowchart 2

Figure A.3: Subflowchart 3
Turtle in trajectory off the ball \rightarrow \begin{align*} & \text{Intercept 12} \\ & \text{Turtle not in trajectory off the ball} \end{align*} \rightarrow \begin{align*} & \text{Intercept 9} \end{align*}

Figure A.4: Subflowchart 4

No teammember behind turtle \rightarrow \begin{align*} & \text{Intercept 11} \\ & \text{Teammember behind turtle} \end{align*} \rightarrow \begin{align*} & \text{Intercept 10} \end{align*}

Figure A.5: Subflowchart 5

Ball is not in the touch line area \begin{align*} & \text{Ball is in the touch line area} \\ & \text{Ball is in the goal line area} \end{align*}

Ball moving towards turtle \rightarrow \begin{align*} & \text{Ball moving away from turtle} \\ & \text{Ball} \end{align*}

Ball velocity low \rightarrow \begin{align*} & \text{Ball velocity high} \\ & \text{Intercept 10} \end{align*} \rightarrow \begin{align*} & \text{Intercept 8} \\ & \text{Intercept 10} \end{align*} \rightarrow \begin{align*} & \text{Intercept 8} \\ & \text{Intercept 10} \end{align*}

Ball velocity high \rightarrow \begin{align*} & \text{Ball velocity low} \\ & \text{Intercept 13} \end{align*} \rightarrow \begin{align*} & \text{Intercept 10} \\ & \text{Intercept 14} \end{align*}

Ball moving towards turtle \rightarrow \begin{align*} & \text{Ball moving away from turtle} \\ & \text{Ball} \end{align*}

Ball moving away from turtle \rightarrow \begin{align*} & \text{Ball} \\ & \text{Ball} \end{align*}

Figure A.6: Subflowchart 6
Ball is not in the touch line area for goal line area

Opponent close to turtle

Opponent not close to turtle

Ball is in the goal line area

Ball is not in the goal line area

Opponent close to turtle

Opponent not close to turtle

Figure A.7: Subflowchart 7