Introduction Lab Humanoid RoboCup

Group Report Team Yellow

Minkai Hu, Fabian Krautmayr, Jingjie Jiang, Tianming Qiu, Zhiyi Li, Yao Rong

Lehrstuhl für
KOGNITIVE SYSTEME
Technische Universität München

Prof. Dr. Gordon Cheng

Supervisor: M.Sc. Mohsen Kaboli
Begin: 24.04.2017
End: 15.08.2017

Abstract

RoboCup (short for Robot Soccer World Cup) is an annually held tournament bringing research teams from all over the world together, to present the latest improvements in the fields of artificial intelligence and robotics. In the practical course "Introduction to Humanoid RoboCup" students get familiar with the humanoid robot NAO and the used framework. Furthermore, the robot's behavior is analyzed and improved by working on a small semester project. The two main projects of team yellow was the kicking behaviour and localization.
An introduction is presented followed by each single improvement made and lastly a conclusion including future work.

Contents

Chapter 1

Introduction

RoboCup, namely "**Robot Soccer World Cup**", is an annual international robotics competition which was founded in 1997. The aim of this game is to promote robotics and AI research in a public appealing but challenging way. In 2017, it was held in Nagoya, Japan, with more than 500 teams from over 50 countries competing against each other. There are mainly five major categories in the whole RoboCup, namely *Robocup Soccer, RoboCup @ Home, RoboCup @ Work, RoboCup Rescue, RoboCup Logistics*. The main domain, RoboCup Soccer, consists of teams using identical robots, Aldebaran Robotics humanoid NAO (since 2009, previously Sony AIBO), playing a fully autonomous soccer game. The teams are normally supported by a university or research institute with strong technical and industrial backgrounds.
The lab course "Introduction Lab Humanoid RoboCup" was offered at the Institute for Cognitive Systems. A team of at least 6 master students was created who focused on different topics and improvements which are more detailed in chapter 2 and 3.

1.1 Motivation

The Master of Electrical and Computer Engineering at the Technical University Munich consists of a two year program in which the students have to choose on one of nine Centers of Competence. One of the center is Automation and Robotics which focuses on topics like Computer Vision, Machine Learning, Humanoid Sensors and Actuators and other robotics relevant topics. The practical lab "Introduction Lab Humanoid RoboCup" offers a big incentive in applying the learned theoretical topics for practical use.
The NAOs offered a big variety of topics the students can focus on, ranging from Computer Vision, Locomotion, Machine learning etc. Being part of a team involved more tasks than just applying technological knowledge: Project management, team organization, team building and other project relevant topics was part of the lab.

1.2 Overview

This summer semester 2017 the practical lab "Introduction Lab Humanoid RoboCup" was offered for the first time. In the previous semesters it consisted of only one lab course: "Humanoid RoboCup" which was split to "Introduction Lab Humanoid RoboCup" and "Advanced Lab Humanoid RoboCup" this semester.

The lab consisted of two parts: The first month consisted of a introduction of the RoboCup and NAO. The introduction part was done with help of our supervisor and a student of the previous semester. We used a wiki and a quick start guide prepared by our teaching assistant Zhiliang Wu

to speed up the introduction part (The quick start guide can be found under A.1-4).
After the introduction the students were assigned to a team (blue and yellow) and each team created subteams for the striker, defender and goalkeeper. Each subteam focused on one major problem which was the project for the remainder of the semester.
The code framework used for the NAOs consisted of a modified B-Human CodeRelease which can be found under (https://github.com/bhuman/BHumanCodeRelease). The B-Human team is the current world champion of 2016 and using their code is a strong baseline for future work. This code was improved in previous semesters.

1.3 Goals

In phase two of the lab a short demonstration was held. In the demonstrations multiple flaws of the robots were detected: The robots have localization problems, the robots have very basic behaviors and other problems. The two addressed problems were localization and kicking strategies.
The tackled localization problem consisted of the following solutions:

- Penalty Mark Perception
- Vertical Line Perception
- Visual Odometry

For kicking strategies solutions were:

- Kicking Methods
- Foot Selection
- Player Alignment

The next chapter focuses on the improvements made.

Chapter 2

Kicking Methods

After the Introduction part a short demonstration was done showing the NAOs with their respective roles. The demonstration shows multiple aspects of the behavior of the NAOs which can be improved. The two aspects improved were the localization and kicking behavior.
These two aspects were chosen for following reason. The striker (subteam for kicking behavior) showed a good performance in the demonstration in that it could localize itself and detect the ball and goals. After the ball and goal detection the striker would go to the ball, align himself behind it and shoot it towards the enemy goal. While the basic behavior of the striker was satisfactory we set the goal in improving the multiple aspects of the kicking behavior. The other subteam focused on the localization problem because the goalkeeper and defender had major problems finding the starting point. To solve the localization problems multiple methods were implemented.

2.1 Roles in RoboCup

To get a better understanding of the actual problem, it is necessary to have a quick look on the different roles that the robots can represent. As it is described in [3], each team in the Standard Platform League (SPL) consists of five players. While one of these is a goalkeeper, the other four players can be defenders, supporters or strikers. In which field areas the players are mainly active can be seen in Figure 2.1. Clearly, the goalkeeper and defender are responsible to protect the goal. The supporters have a hybrid role, where they are more defensive or offensive, depending on the current situation. Finally, the striker is in charge of scoring and mainly active in the opponents half.
Since the RoboCup team at TUM is at the very beginning of it's development, the complex role of the supporter is not implemented yet. Therefore, we only use three players, namely goalkeeper, defender and striker. This means that the striker is covering a much bigger area than usually in the SPL (see Figure 2.2). Hereby, the original behavior of these three roles was created by the RoboCup team called B-Human [16]. During this and also the prior semesters, this behaviors got improved by students from TUM.

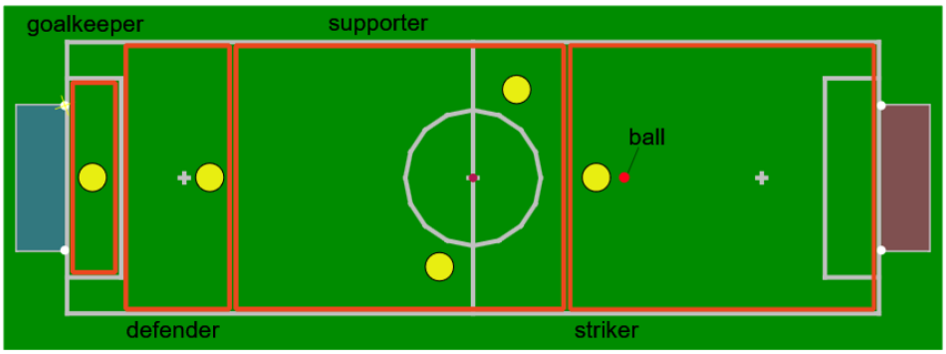


Figure 2.1: Roles in SPL

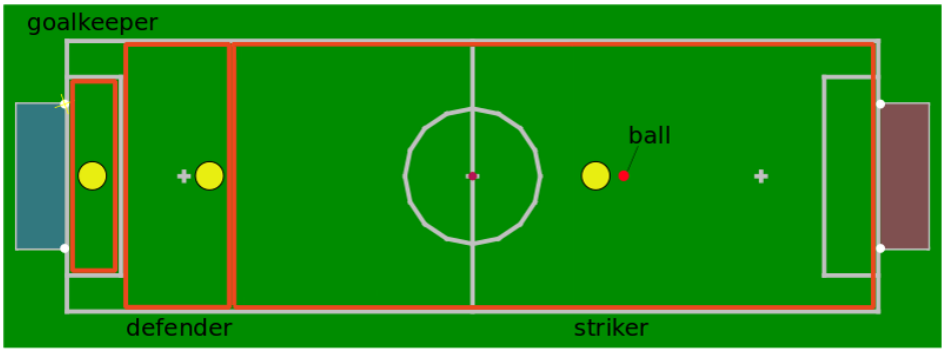


Figure 2.2: Roles at TUM

2.2 Kicking Methods

In this section the focus lies on the implementation of "kicking methods". First a motivation for the implementation is presented followed by the implementation details and results

2.2.1 Ground truth of kick

The win condition in soccer is to score a goal. To score a goal the striker or sometimes the supporter has to shoot the ball into the goal while avoiding getting caught by the goalkeeper. A penalty kick underlines the importance of the kick: Does shooter go for a strong but uncontrolled kick or a weaker but more precise kick. One has the merit of being more difficult to block while increasing the possibility the goal while the other one can target more difficult positions of the goal but allows the goalkeeper enough time to react to the shot. Every professional soccer player mastered many kicking methods, so if the aim of RoboCup is to play against the real world champion in soccer by 2050 mastering multiple kicks for different situation is of utmost importance.
The striker in the demonstration showed a very basic kick as it would always perform the same routine: localization, ball detection, alignment to the goal and performing the kick, repeat. This presents multiple problems: one of the problems was the same kick procedure for every possible condition. The striker does not discern between the location of itself in the field or the enemy goalkeeper/players. It always performs a weak kick resulting in a ball movement of around 20 to 100 centimeters. With a field size of 10.4 meters by 7.4 meters (3.2 meters in our case) the robot needs many kicks to get the ball close to the enemy goal while the enemy has a lot of time available to intercept and steal the ball for a counterattack. Another with a single kicking method is that if an enemy stands between the ball and goal it results in the guaranteed interception of the ball, due to the fact that the kick is always targeted at the goal. In a real soccer game the players have different tactics to circumvent a defender trying to block: for example a kick to the sides. The improvements made are the implementation of different kicking methods of different kicking strength and direction. This improvement is combined with the improvement of foot selection (Chapter 2.3). A video of the original kick can be found under `https://youtu.be/7frVjvR8t9k`

2.2.2 Improvements

The process of a kick consists of multiple smaller motion sequences. A newly flashed NAO needs a input sequence of each limb to perform a kick. To perform this sequence multiple conditions need to be met: First the joint limitations of the NAO has to be considered. The inputs of e.g. the left knee pitch must range between -5.29 and 121 degrees (In [1] the full limitations of all joints are listed). Another conditions is that the sequence is stable on each sequence, that means it should be staying in its pose whenever the sequence is stopped[10]. The center mass should always stay horizontally on the support area 2.3.

The results consisted of a series of commands (phases) in a .kmc file. Each phase was part of the sequence with a defined duration in which time the robot would need to change the joints to the respective values. In each phase every joint has a target xyz value it has to reach. While in the simulation SimRobot the conditions are perfect the conditions for the NAO are not perfect, disturbances hinder that real values are the same as the given values. This is counteracted by using a PID controller with importance laying in the proportional gain (Because correcting the current error has the highest priority). A small snippet of the kmc file can be found in A.5
To improve the kick an analysis of real soccer players performing different kicks had to be made. A stronger kick is performed by using a higher velocity of the foot when hitting the ball. While the kicking direction can be varied by using the inside edge or outside edge or the rotation of the foot. The main part of the kick is the high angle of the knee when winding up and an almost

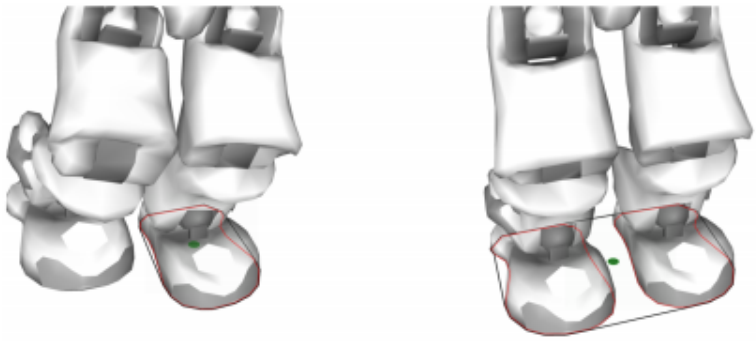


Figure 2.3: Center of mass condition for stability [10]

straight line after hitting the ball (see Fig 2.4). A stronger kick was implemented by reducing the overall process duration and increasing the wind up range and range of the follow through. While an angled kick was created by changing the rotation of the feet and body while keeping the balance.

Figure 2.4: Angle of feet when performing a kick [8]

Figure 2.5 shows the previous kick on the left and the improvements on the right side
With the additional kicking options the striker could kick the ball from the kickoff position into the enemy goal (up to 5 meters). This improvements allows a much bigger area from which scoring a goal is possible. While the angled/directed kick is more of an advanced tactic it is the one used most commonly in a soccer game, mainly for passing and avoiding the enemy team. A video of some of the kicks can be seen under `https://www.youtube.com/watch?v=0IADRnFe5QA`. These kicking methods were combined with the subsequent improvement, the foot selection.

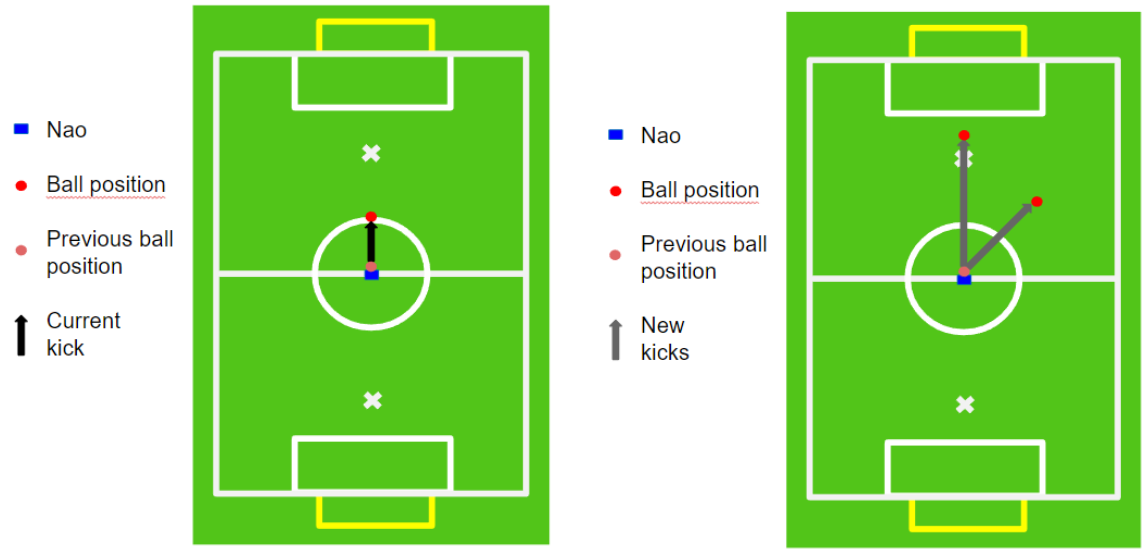


Figure 2.5: Previous kick (left) and proposed improvements (right)

2.3 Foot Selection

In this section, we focus on the implementation of the project that is called "Foot Selection". Thereby, we discuss why such a method is necessary and how it can be realized. Furthermore, we describe the implementation of the foot selection into the already existing code of the striker.

2.3.1 Problem Definition

Regardless of whether we have humans or robots on the field, soccer is a game where a quick execution of actions can be very important. Thereby, one of the most used actions is the kick. That is why a good kicking performance is essential to succeed in the RoboCup competition.

While working with the NAOs during the introduction phase, we figured out that the robots are always kicking with the left foot. This can be also seen in the left flow chart in Figure 2.7. This particular behavior does not cause any problems, if the ball lies closer to the robot's left foot than to it's right foot, because the process of aligning behind the ball can be solved very quick. But if the ball lies closer the the robot's right foot, the NAO needs several extra seconds to align itself behind the ball such that it can kick the ball with the left foot. These two cases can be also seen in Figure 2.6. While for the situation on the left hand side a kick with the left foot is suitable, the better choice in the situation on the right hand side would be a kick with the right foot.
In a game situation, these seconds can decide whether a defender is able to block a striker's scoring attempt. That is the reason why we have seen a huge need to give the NAO the ability to decide which foot to use, to ensure a fast execution of the kick action.

2.3.2 Proposed Solution

To solve this problem, it is necessary to calculate which foot is closer to the ball. This can be done by using the NAO's field position on the one hand and the ball's field position on the other hand. With these two parameters, we can determine the ball's relative position to the robot and therefore derive which foot to take.

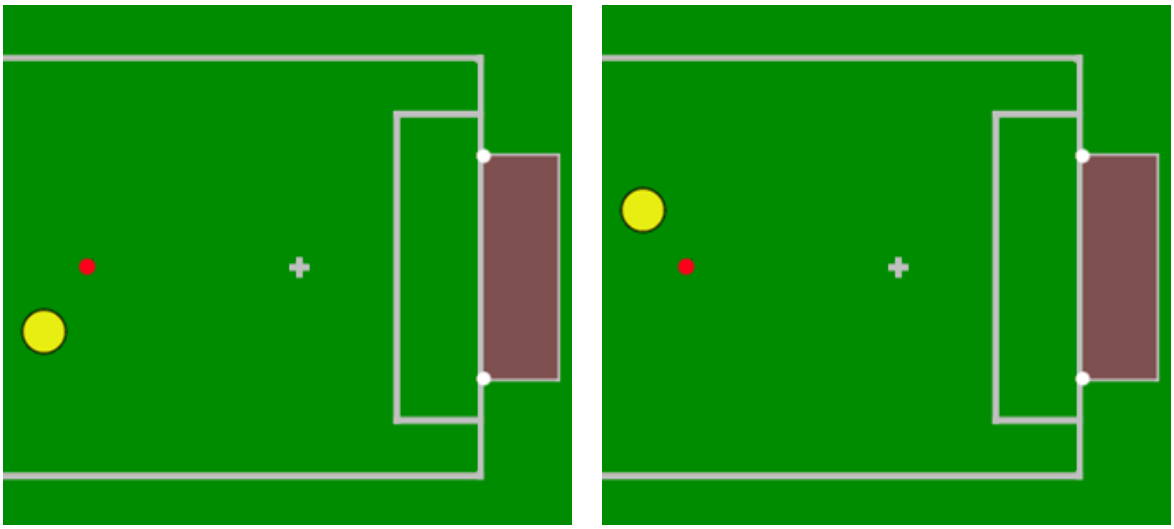


Figure 2.6: The two cases for foot selection

As we can see in the right flow chart in Figure 2.7, this process happens after the striker turned itself to the ball. Therefore, a new state called “Foot Selection” got introduced. After passing through this state, the robot can smoothly walk to the ball and afterwards do the alignment process. Thereby, the robot aligns itself behind the ball either with the right or the left foot, depending on the outcome of the foot selection. Last but not least, the kick gets performed with the chosen foot.
With this method, the striker can now kick with both feet and saves about 2 to 3 seconds during the alignment process. This increases the possibility of scoring, since the defending team’s robots have less time to react. A video which demonstrates the foot selection process can be found under following link: `https://youtu.be/inE38jgFDSE`.

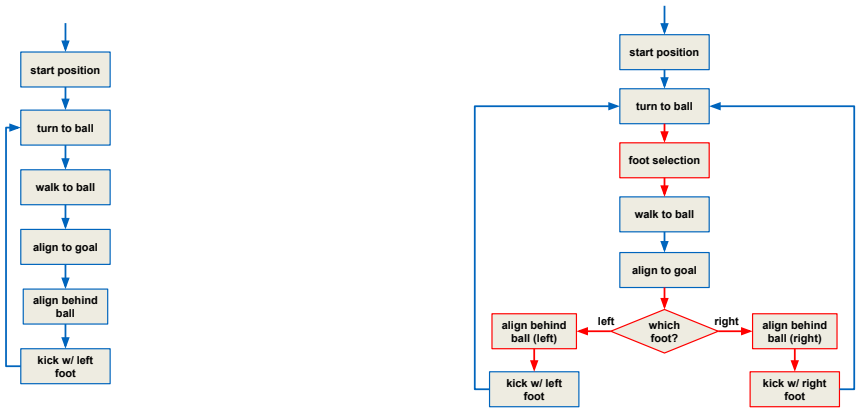


Figure 2.7: Striker: flow chart of the ground truth (left) and dynamic kicking direction (right)

2.4 Improved Alignment to the Goal

2.4.1 Problem in alignment to the goal

In both simulation and real environment, the striker kicks the ball to his own goal, when he locates himself between the ball and the goal. (please see in `https://youtu.be/mwRH3JHcdVM`) The reason for that is that, while the striker is aligning himself behind the ball after finishing the alignment to the goal, he does not know that he is kicking the ball to his own goal. In this case, he cannot get rid of the terrible situation. Consequently, he cannot score any more.

2.4.2 Proposed solution

To solve this problem, alignment to the goal is modified. As shown in Figure 2.8, when the striker locates himself between the ball and the goal, he should differentiate between two cases: whether he is on the left side of the ball or on the right side of the ball. For both cases, the improving idea is the same and consists of three steps. We take the case of standing on the left side of the ball as example. The striker firstly walks along the positive y axis until he is 250mm away from the ball. Then he walks along the negative x axis until he is 300mm away from the ball. Afterwards, he aligns himself to the goal as he usually does. This strategy guarantees that the striker avoids kicking the ball to his own goal when he locates himself between the ball and the goal, which has an indirect positive effect on scoring. A demonstration video for the improved alignment to the goal is located in `https://youtu.be/c6ZrAf32stU`.

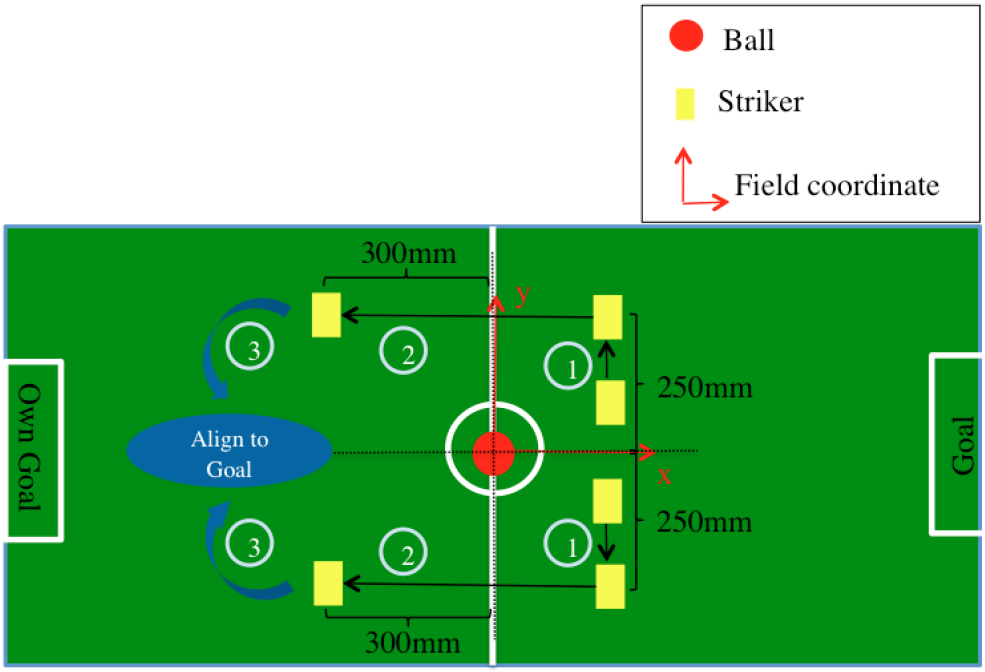


Figure 2.8: Improved Alignment to the Goal

2.5 Area-based Alignment

2.5.1 Problem in the field corner

In the soccer game, the chance to directly score in the field corner is normally relatively small, since the opponent goalkeeper might block the ball or the ball might be kicked outside the field. As a result, the striker will lose the ball. Figure 2.9 shows that the striker insists on scoring directly in the field corner, highlighted in blue. Because he has no other choice.

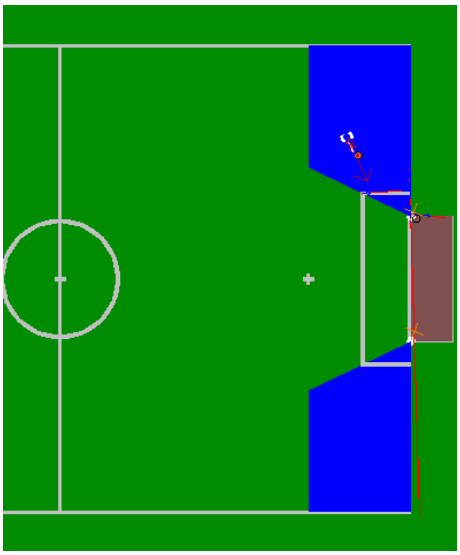


Figure 2.9: Alignment to the Goal in the Field Corner (Adapted from [11])

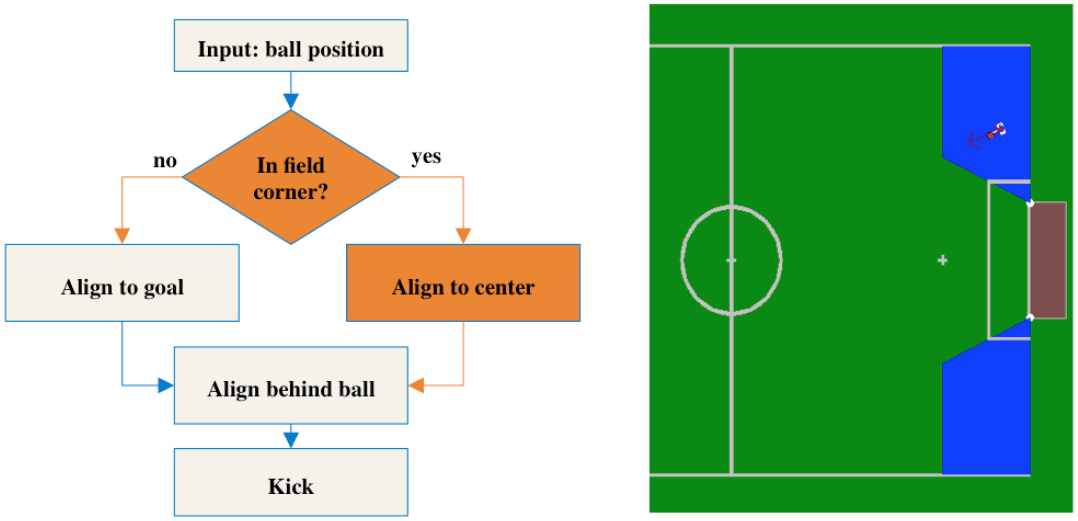
2.5.2 Proposed solution

When the striker locates himself in the field corner, it is better to kick the ball towards the center. Based on that, another variant of alignment is added, namely alignment to the center. The flowchart in Figure 2.10 (a) shows that the striker will align himself according to the ball position. When the ball is in the field corner, the striker aligns himself to the center and kicks the ball to the center, otherwise he aligns himself to the goal and kicks the ball to the goal. One could see the success of the striker's alignment to the center in the field corner in Figure 2.10 (b). There are two main advantages of the idea of the area-based alignment:

1. The possibility of scoring could be increased with kicking the ball to the center in the field corner.

2. It builds the foundation of passing strategies, i.e. different kicking directions.

A demonstration video for the area-based alignment is also provided in `https://youtu.be/CLicXGWBU9Y`.



(a) Flowchart of the Striker Behavior with Area-based Alignment

(b) Alignment to the Center in the Field Corner

Figure 2.10: Area-based Alignment

2.6 The Skill of Dribbling

In this project, we used the dribbling strategy in [18].

2.6.1 Concept

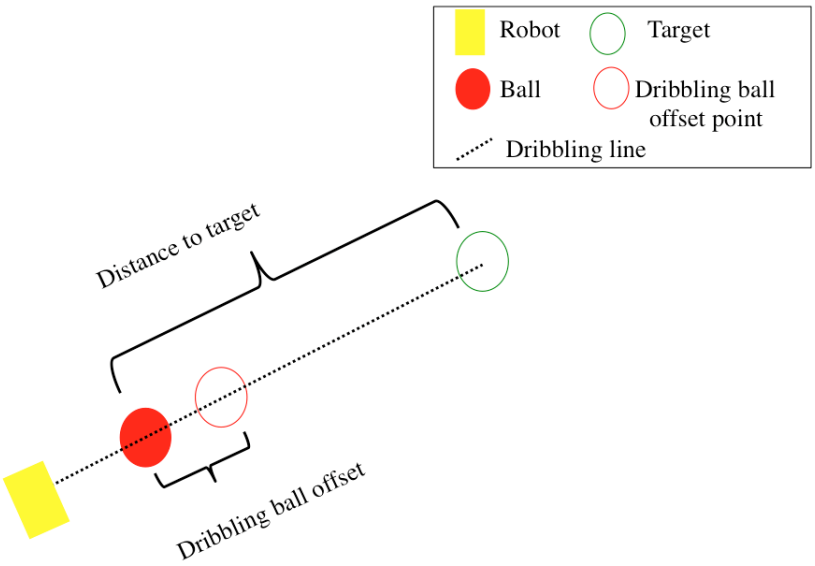


Figure 2.11: Concept Of Dribbling (Adapted from [18])

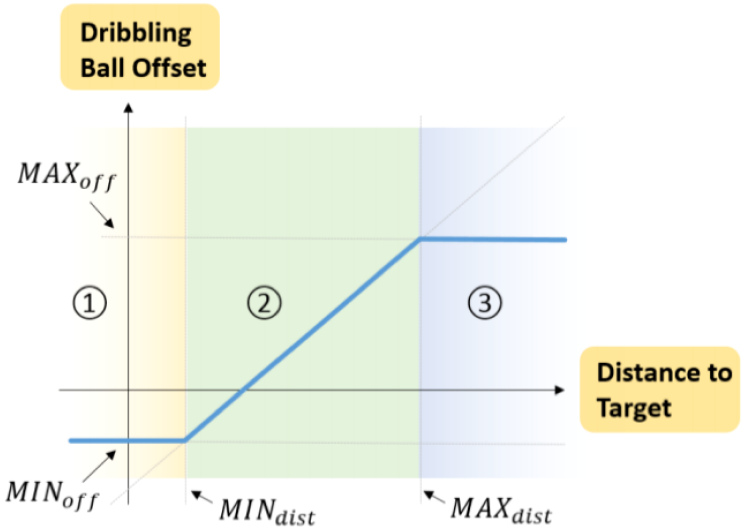


Figure 2.12: Distance Dependent Dribbling Offset [18]

The main idea of the dribbling strategy is to let the striker walk the ball to a predefined target position along a desired dribbling line. To achieve this goal, in each iteration a dribbling ball offset is set, shown in Figure 2.11, until the target is reached. Due to the proportional relation between the dribbling ball offset and dribbling speed, the striker could better keep the ball close to himself with small dribbling ball offset, while he might kick the ball out of his control with large dribbling ball offset. Related to this fact, there is a trade-off problem between dribbling speed and ball controllability. A function describing the dependency of the dribbling offset on the distance to the target is introduced in Figure 2.12 to tackle this trade-off problem. It consists of three parts. When the ball is still far away from the target, larger than MAX_{dist}, the dribbling ball offset is set to the maximum threshold MAX_{off}, since it is desirable to rather roll the ball as soon as possible to the target. On the contrary, having a stable control on the ball is more important near to the target, smaller than MIN_{dist}. For that, the minimum dribbling offset threshold MIN_{off} is used. In case of between MIN_{dist} und MAX_{dist}, the dribbling ball offset is calculated by using a linear relation to the distance to the target.

2.6.2 Simulation result

We programmed the dribbling steps introduced in [18]. In the simulation environment, the dribbling does not perform as expected. At the beginning of dribbling, the striker walked the ball along the dribbling line. Later on, he unexpectedly deviated from the dribbling line regardless of the ball. Because of the timeframe of the project, we failed to dig into reasons for the unexpected dribbling behavior. However, we would suggest two possible reasons, which might be helpful for the future project on dribbling:

1. The parameters overall are not adjusted properly.
2. There are errors in the code presenting the dribbling algorithm.

Chapter 3

Self-Localization

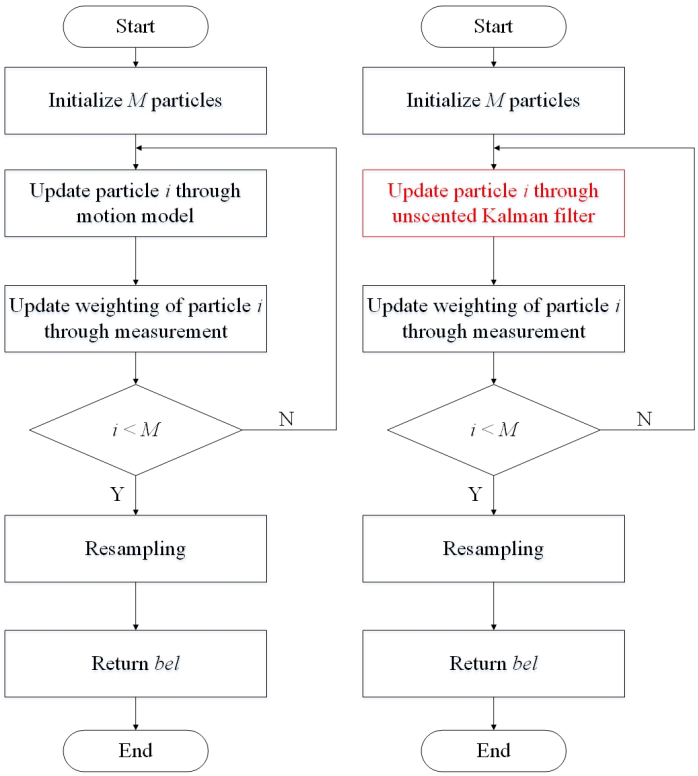
This chapter introduces the penalty mark and line perception improvement and the concrete method used in the current framework for NAO, the mobile robot localization. RoboCup self-localization is a problem that NAO robot could determine its pose, which includes both position and angular, relative to the given map of the standard soccer field. Almost all the tasks for each robot player require knowledge of its pose on the field. We call this problem as rather a self-localization than a SLAM(Simultaneous localization and mapping) problem, one important reason is the map of the environment has been provided as prior knowledges. Self-localization could be treated as a problem of coordinate transformation. Map of the field or environment is described as the global coordinate system, which does not rely on robot's pose. The robot pose will be finally transformed into the global frame and represented as a global coordinate[19]. In our project the NAO pose is only changes in a plane, the z element is always remains as zero since the robot always stands firmly on the ground. So the pose could be written as $(x, y, \theta)^T$.
However, the pose of robot could not be acquired directly, instead it could be inferred from the sensor data, which here we read it from cameras. The sensor could not be hundred percentage accurate, it always contains noise, especially image processing by using camera or illumination changes or other possible visual noises. So we have to find a proper method to calculate the pose from imperfect sensor data, which will be introduced in this chapter.

3.1 General Framework: Particle Filter

For self-localization, B-Human uses a particle filter based on the Monte Carlo method [6] as it is a proven approach to provide more accurate results in such an environment [17]. The reason for selecting particle filter in this self-localization problem is that in the RoboCup soccer match the robot will from time to time fall down, and then it will not get the correct knowledge of its position. The issue is that the robot might believe that it knows where it is while it does not. This is called "kidnapped robot problem" [19]. So the particle filter will be adopted to solve this kidnapped robot problem and it does not demand initial position of robot.
The particle filter is a kind of Bayes filter, based on Monte Carlo method. It uses several samples as particles to describe the posterior. In particle filter, the samples of a posterior distribution are called particles are represented as [19]:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]} \dots x_t^{[M]}$$

The posterior of each particle $x_t^m (with 1 \leq m \leq M)$ is the probability of the state at specific time t. Usually, M, the number of the particles is very large to get a as perfect as possible approximation, since particle filter is a non-parametric method to estimate the state updating. However in the RoboCup situation, only 12 to 16 particles are adopted. Because the memory



(a) Particle filter (b) combine PF and UKF

Figure 3.1: Self-localization based on particle filter flowchart

space and computation speed are both limited on the robot NAO. During the match, it has a high demanding on real time performance. Here I have not tested or verified the effectiveness of the so small number of samples. But in recent years B-Human team even drop the particles from 16 to 12 [15] and it is proven in practice which shows a great robustness and outstanding performance. Then how does robot NAO apply particle filter to localize itself is shown in the flowchart Figure 3.1(a).

3.2 Combination with Unscented Kalman Filter

Since 2012, B-Human framework made a big change on self-localization: a combination of a particle filter and an Unscented Kalman filter is used. The former computes a global position estimate; the latter performs local tracking for refining the global estimate [14]. This implementation will improve the local accuracy of particle filter and further results in the improvement of localization precision. The change has shown in the Figure 3.1(b) .
The idea behind the Unscented Kalman filter is similar to the original Kalman filter. However it is applied for the non-linear motion model and provides preciser result than Extented Kalman filter which only uses first order approximation of Taylor series. The idea of Unscented Kalman filter is that it generates sigma points and uses the unscented transform to describe the non-linearity. The mean value and covariance calculation is based on these sigma points. Since the state of pose could be represented as $(x, y, \theta)^T$, it contains three dimensions, then the number of sigma points for each state is $2n+1 = 7$, here n is the dimension number of the state variable. Each sigma points will be predicted by motion model. And due to the non-linearity of motion model, unscented transform will be used. As shown in Figure 3.2, the mean as well as covariance of states will also be modified

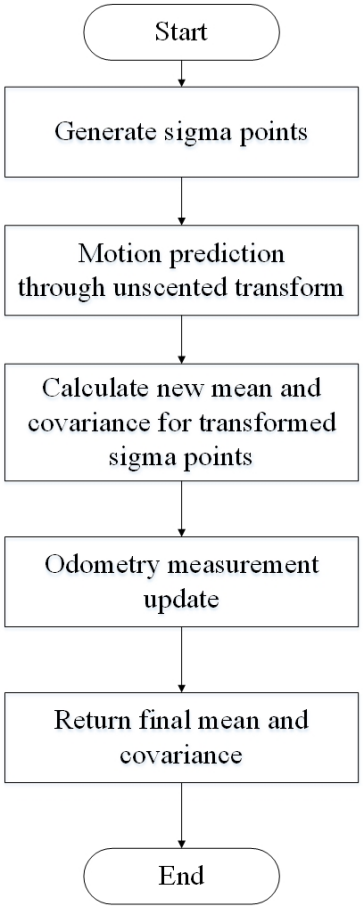


Figure 3.2: Unscented Kalman filter

with measurement and finally be returned.
Generally, the localization process will use particle filter to estimate the robot pose and each particle will use unscented Kalman filter to update the pose of the sample.

3.3 Penalty Mark Perception

3.3.1 Current problems

To determine a spot is on penalty mark or not, there are many criteria to discard those line spots that is not satisfy the property of penalty mark. The current method and problems of this method will be discussed in this section.
Line detection and penalty mark detection both need the process of detecting the possible line spots. In B-Human code release 2015 introduced the criteria to evaluate line spots that from a penalty mark. The criteria are as follows:

1. Could it from a penalty mark?
2. Is it far enough away from the field border?
3. Is it close enough to the observer?
4. Is it not inside a ball?

The ball in real football game is black and white, the white spots on football may recognize as a spot on penalty mark, it will be a problem for the penalty mark perception. But for Robocup we used the red ball, the last criterion can be ignored. A penalty mark is always far away from the field border. The second criterion make sure that the spot won't be recognize as a spot may on penalty mark, if the distance of the spot to the boundary is too small. If the distance of a spot from the robot is beyond a certain value, the spot is discarded because it is most likely a false positive. The first criterion includes many small criteria for example, whether is the size of the detected spots fits penalty mark, whether is the variance of those checked spots below a threshold.[16]

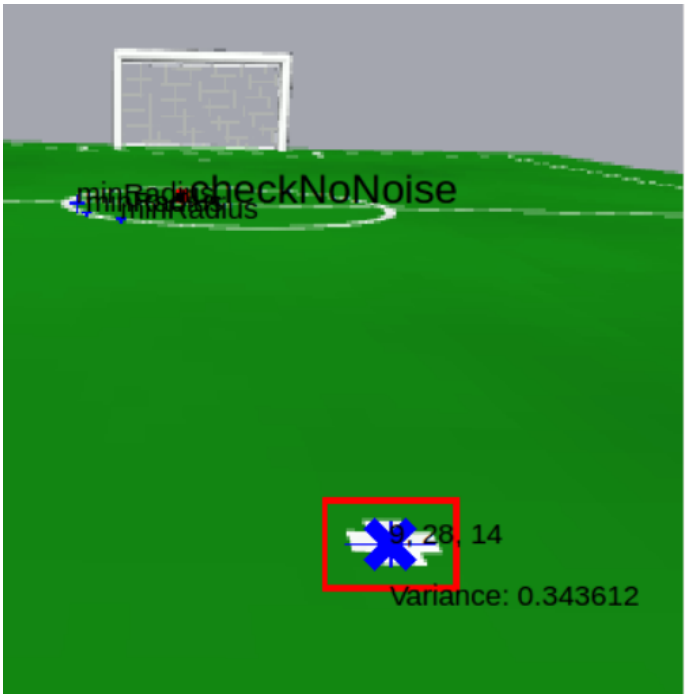


Figure 3.3: A recognized penalty mark

Two problems were found in experiment:

1. A good penalty mark can't be recognized as a penalty mark
2. Variance is negative

A well-detected penalty mark can't be recognized, one possible reason is the variance is beyond the threshold (the threshold is 40).
To find a reason for the first problem, the scene was simulated in SimRobot. In simulation, another problem shows.

Table 3.1: seven tests and variance record

	1	2	3	4	5	6	7
variance	-0.33697	-0.661187	12.9169	-0.69982	44.29901	-0.485818	0.190308

Besides the first problem, the table shows another problem is that the value of variance sometimes can be negative. A negative variance is impossible in mathematics. The variance calculation may have problem.

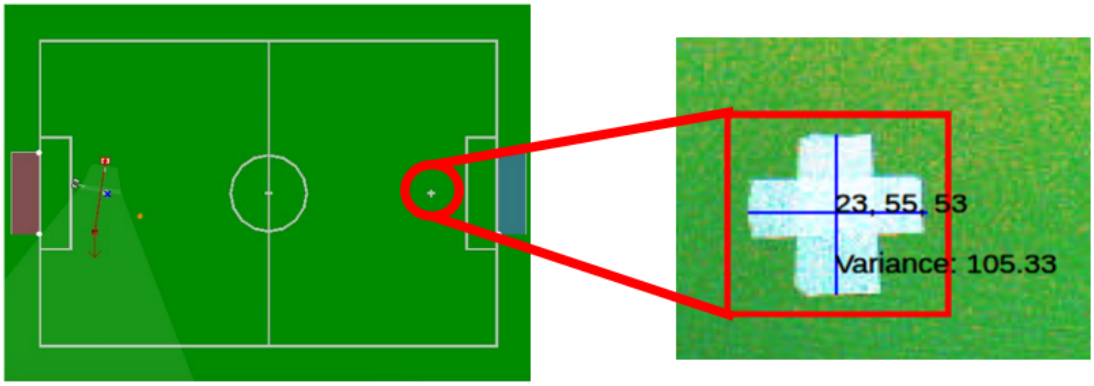


Figure 3.4: A good spot cannot be recognized as a penalty mark

3.3.2 Current method

The Method that B-Human used to calculate variance called naive algorithm. To compute a sample variance, the fundamental calculation consists of computing the sum of squares of the deviation from the mean,

$$S = \sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{3.1}$$

where

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.2}$$

With above two equations a so called one-pass algorithm can be derived:

$$S = \sum_{i=1}^{N}(x_i)^2 - \frac{1}{N}(\sum_{i=1}^{N} x_i)^2 \tag{3.3}$$

This equation (3.3) is mathematically equivalent to 3.1, but with floating-point arithmetic it will be a disastrous. The quantities in 3.3 may be very large in practice, and will generally be computed with same rounding error [2]. If the input number to large an overflow in floating-point arithmetic may change the symbol of the result. A negative variance is in mathematically impossible. Thus, this algorithm should not be used in practice.

3.3.3 Welford's method for computing variance

In the first part of this chapter introduced a method to solve the problems in section current problems. This algorithm also is also written in the codes. The second part shows the result with using the new method.
Instead of naive algorithm, the Welford's method can be used to avoid the disadvantages of the naive algorithm and improve the method to computing a variance. Welford's method updating the sum of squares with the current mean, after computing every time the variance, the data do not need to be saved for a second pass. Welford's uses following formula to compute sample variance[9]:

$$M_{2,n} = \sum_{i=1}^{n}(x_i - \bar{x}_n)^2$$

$$M_{2,n} = M_{2,n} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

$$variance = \frac{M_{2,n}}{n-1}$$

The $\bar{x}_n$ is the current data mean and the $\bar{x}_{n-1}$ is the data mean from the last iteration. Welford's method performance better in accuracy and in extreme circumstance compare with the naive algorithm.
To compare with Welford's method, the standard way to computing variance was tested. The standard variance calculation is:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$variance = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

The result how those two methods performed will be introduced in next section.

3.3.4 Result

The following results were collected in the same situation. It means that in every test the robot has the same start position and the same walking trajectory.

Table 3.2: seven tests and variance record

	1	2	3	4	5	6	7
standard	0.10183	0.103699	18.0895	6.03105	19.0168	12.2954	0.101453
Welford	0.2572	0.2486	0.2526	0.251778	0.3914	0.3966	0.24897

In seven trials, when the robot passed by the penalty mark and the penalty mark was within the field of view, it can be recognized and marked as a penalty mark. The result in above table show that these two methods can solve the negative value problem. Compared with the standard method, Welford's method has more stable and smaller result. This method guarantees those good spots won't be discarded by erroneous calculation of variance.
Because the first step for penalty mark perception is line spots perception, the implementation in the next section can also improve the penalty mark perception.

3.4 Line Perception

The white lines on the field are important features. But our robot had problem to "see" the lines with very large slope, especially the vertical lines.
The process of line perception can be explained as in the Figure 3.5.

"Two images" are the images captured by two cameras, namely upper and lower camera. The improvement could be found in each of the rest of the states. In the following sections, the details about the scan grids, the condition for validating edges, finding the potential line spots and drawing the lines will be explained.

As the Figure 3.6 shows, there is no lines but just some noise at the right top corner could be detected (marked as cross).

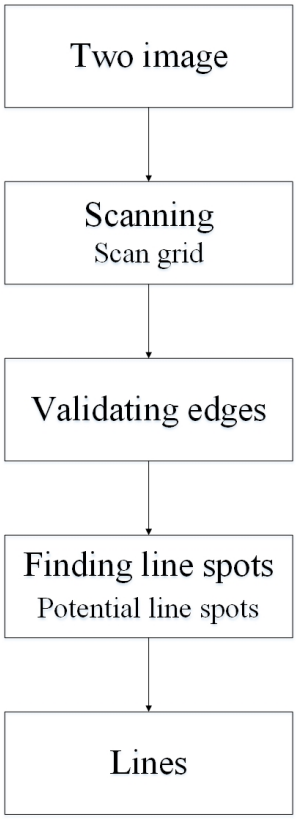


Figure 3.5: Line perception

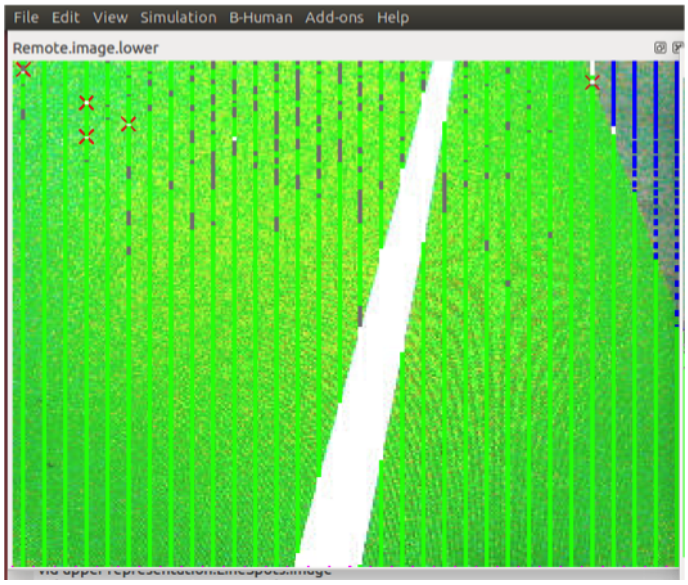


Figure 3.6: No line is detected!

3.4.1 Scan Grid

The tiny green line in the figure is the scan grid. It scans the whole image from the top to button of our image to detect the color changes. After scanning we can get color segments along the scan line. Each color creates a segment. Shown in the Figure 3.7, the red arrow is the scan direction. Moreover, the background out of the field will be "clipped" [13], so we don not have scan grids outside the field.
The scan grid to a reasonable value method will be adopted, that means we get more segments without causing too much burden for our robot's CPU.

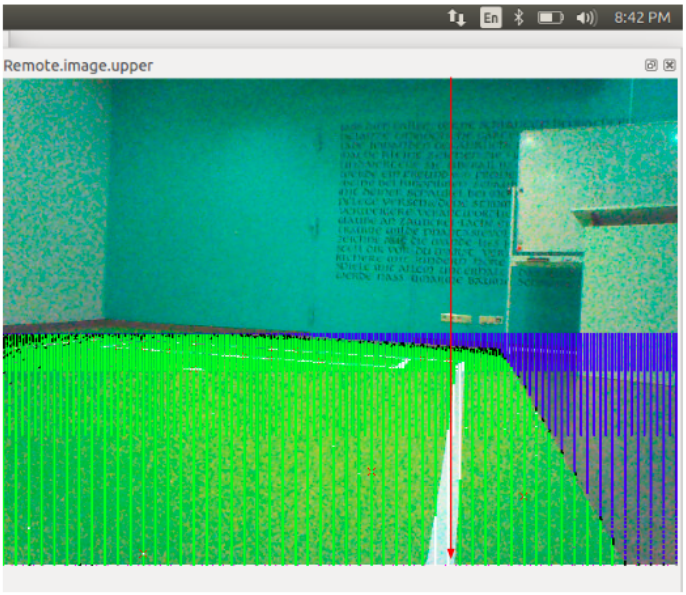


Figure 3.7: Scan grid

Now we can explain why it is not possible for our robot to see the very vertical line. Because there is no color change along the scan line at all. There will not be any segment for the further process. But fortunately, our robot moves all the time, so the vertical line will not be permanent impossible to detect.

3.4.2 Validating the Edges

For the large-slope lines, we need to first make the segment validated. If the white color possesses too much ratio along a grid line, it will not be regarded as a potential line region but a white region belonging to robot or other noisy area. The explicit algorithm about the *Region-building* can be found in **BHuamnCodeRelease2013**.
So we need to set a threshold for the large-slope lines, because their *width/length ratio* are different from the lines with the smaller slope. The following Figure 3.8 shows that, the edges can be detected which are marked as blue crosses. The red crosses are the center of mass of the white regions, which will be used to build the regions as the potential line spots.
The idea of the *Region-building* algorithm is putting the neighbor (white color) segments together as a list in C++. After that we have lots of regions, then we need to cluster them and find the potential line spots, in order to build our lines.

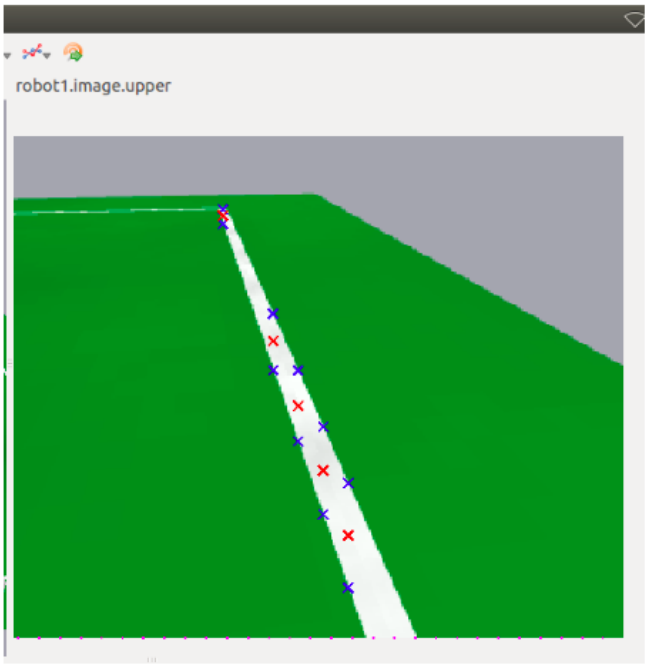


Figure 3.8: The validated edges for the segments

3.4.3 Potential Line Spots

The potential line spots (they are all white regions) we get from the last step should be clustered as different classes, for instance, noise, circle, line, robot body, etc. The criteria for line cluster are for example the difference of directions, or the distance which is calculated by using the Hesse formula. They are explained in the algorithm *Cluster LineSegment* in **BHuamnCodeRelease2013**.
In some situations, we get a lot of noise that we will abandon. But maybe inside those noises, we could still find some "lines" as the Figure 3.9 showing:

Inside the red circle, they are all the spots recognized as "noises". At the right top corner, there is still a line (marked in red). A good method to cope with this is implementing RANSAC.
RANSAC is widely used in computer vision branch, and it is a general parameter estimation approach which is designed to cope with a large proportion of outliers in the input data and uses those white points to extract one line [4].
We implement RANSAC to the noise cluster, when the amount of noise is quite large. Two points are randomly picked up to build a line. When at least four of the rest of points fit the line, then we will put them into line cluster.

3.4.4 Drawing the Line Model

The conventional method to build the line model in 2D coordinate is using the form $y = ax + b$, which causes problem if the slope a is too large (over-floating).
To avoid calculating this large value, we set a threshold for slope a. If it becomes too large, we put the first and the last line spot in this line cluster, which means we set their x position values as the same.

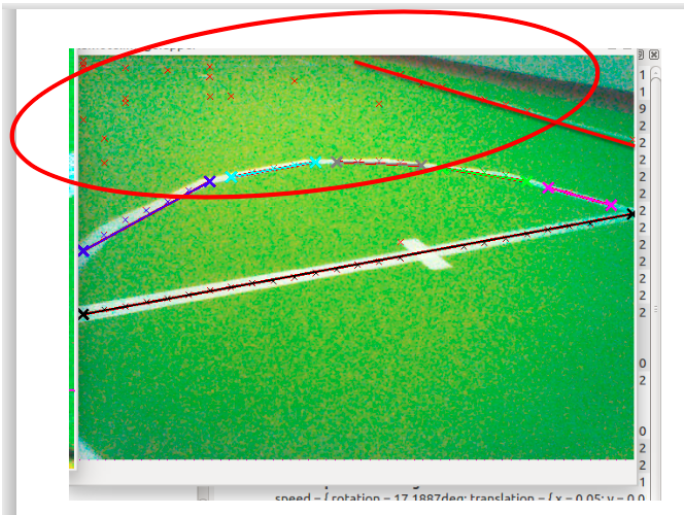


Figure 3.9: The possible line lying inside the noises

3.4.5 Experiment Results

After the improvement, we could achieve our line perception for large-slope lines. The red line in the Figure 3.10 is the model we get. Compared with the Figure 3.6 before, the line perception is obviously improved.

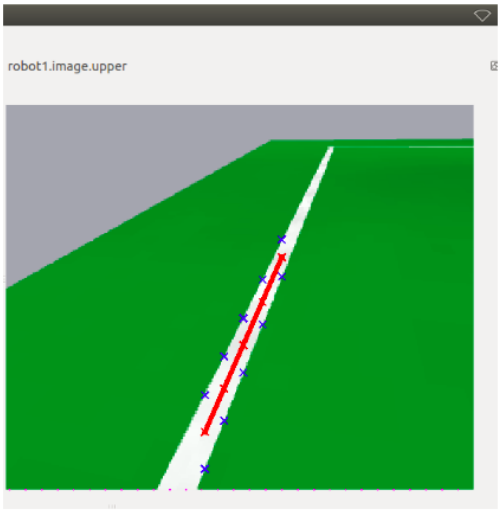


Figure 3.10: Detection of line with large slope

The final task for the project was to implement the improved line perception to NAO robot. The result we got as below:

1. The amount of detected line spots increases two times.
2. NAO (as goalkeeper) reaches his position in the goal area insides $1'40''$ in the simulation (SimRobot).

3. Implementation on real robots. The best result we get is identical with the one in the simulation.

The Figure 3.11 shows us in both scenes. Our goalkeeper achieved to find his place from the game beginning using about one and a half minutes.

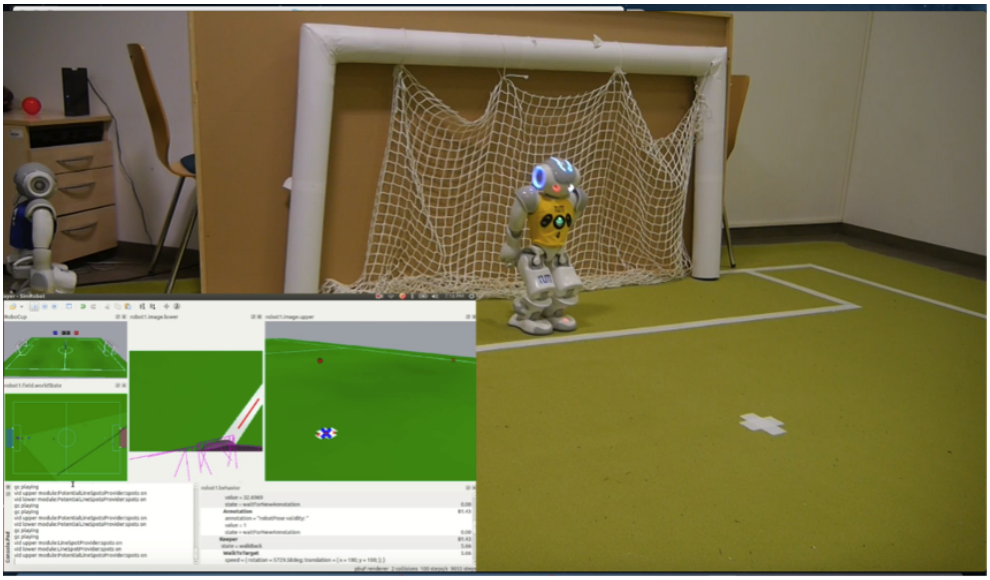


Figure 3.11: Goalkeeper in SimRobot and on the real field

3.5 Visual odometry improvement

3.5.1 Current method

There are several sensors to acquire information from the environment on NAO: camera, microphone and sonar. From them only the camera is adopted, since the environment of the soccer match will be quite complicated, for example:

- The white border lines and green field are on the same plane.
- The goal posts are perhaps too far away from the robots sometimes.
- There are 6 moving robots on the field, which the opponents are distinguished from our teammates only by colors.
- The ball is small and the detection of a ball requires high precision.

So based on above reasons, the another sensors are not qualified for the requirements. However, the camera is suitable for each specific task and the computer vision skill as well as visual odometry are being perfect so far.
There two cameras on the NAO's head which can provide large FOV. Several tasks such as image preprocessing, feature detection(including: line perception, penalty mark perception, ball perception) and data association will be performed on the image gathered by each camera. B-Human's current vision system provides a variety of perceptions that have all been integrated into the sensor model: goal posts (ambiguous as well as unambiguous ones), line segments (of which only the

endpoints are matched to the field model), line crossings (of three different types: L, T, and X), and the center circle [14]. These features can be associated to the feature detected in pixel images. The detailed of these tasks will not be mentioned in my part of report, which are introduced by my teammates. My work is that suppose the the feature detection and data association have been finished.
Based on feature detection and data association, the corresponding point pairs have been already found. For example the goal post in figure i. whose coordinate in the homogeneous global 3D world could be represented as $(X, Y, 0, 1)^T$. And the corresponding homogeneous coordinate in pixel coordinate can be written as $(x, y, 1)^T$. Since the camera calibration matrix, including both intrinsic and extrinsic calibration parameters, has been acquired through camera calibration before each match. Assuming the camera model as pinhole camera model shown in Figure 3.12 [7]:

$$\mathbf{x} = \mathbf{K} \cdot \mathbf{X}_{cam}$$

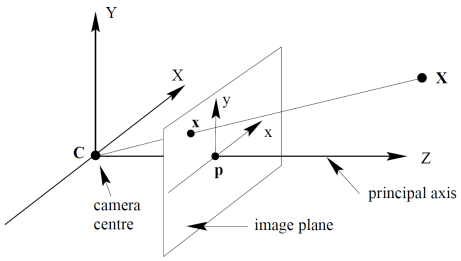


Figure 3.12: Pinhole camera model

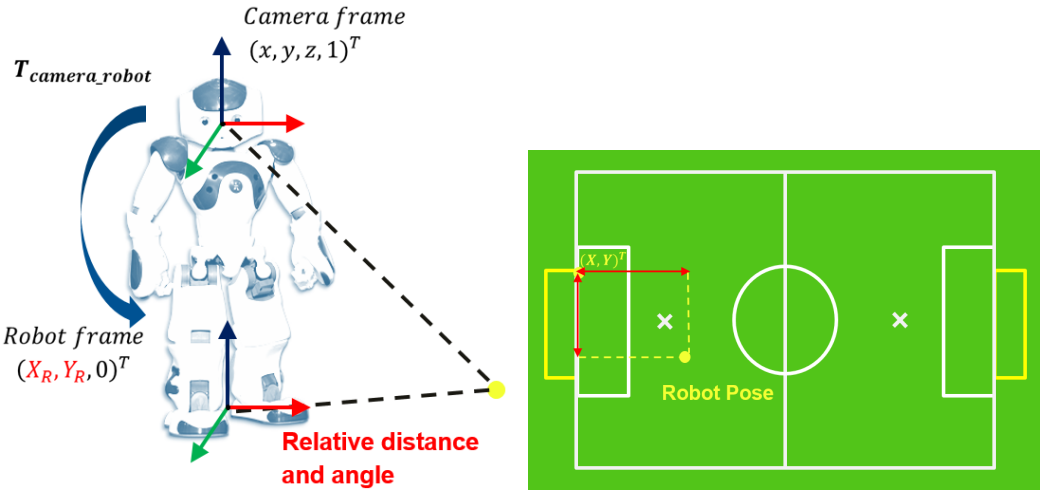
So we could get the corresponding 3D goal post coordinate in the ideal camera frame. Since we know the geometrical value of NAO's body construction, we can transform the coordinates into the robot frame, which is shown in the Figure 3.13(a). Based on this, we could get the related distance between robot and the goal post in robot frame, which is shown in the Figure 3.13(b). At the same time, the precise location of goal post in the global field is also known. So we could calculate the robot pose from this above information.

3.5.2 Current problem

In the process of transforming the coordinate from camera frame into robot frame, the current method only use a constant transform. However when robot is walking, it cannot always be extremely vertical to the ground. The central line of its body and camera on the head will have an arbitrary angle of tilt. Also due to detecting of the environment, robot always turn around its head or even look up and down, but this information will also create bias by coordinate transforming. Then it will cause a bias on the camera frame transform, which is shown in Figure 3.14. This error between the real robot base coordinate and the result calculated by the constant transform will further affect the pose estimation.

3.5.3 PnP Pose estimation

Since the current method contains the tilting and shaking error PnP pose estimation method would be adopted [5], which could eliminate the error by considering the transform in a more direct way. PnP means "Point n Point", which uses n pairs of corresponding point to calculate transform relationship. Reconsidering the corresponding point pair $\mathbf{x} := (x, y, 1)^T$ in pixel coordinate and



(a) Coordinate transform (b) Relative pose to landmark

Figure 3.13: Pose estimation

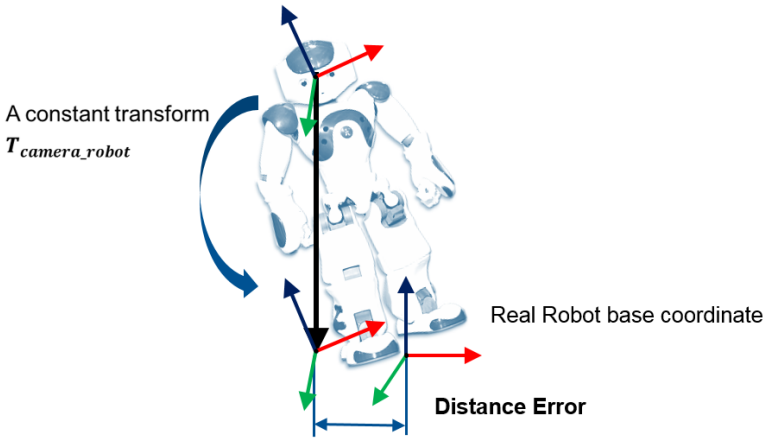


Figure 3.14: Position error by a constant transform

$\mathbf{X} := (X, Y, 0, 1)^T$ in 3D global space. They should have this following relationship (3.4):

$$\mathbf{x} = \mathbf{T}_{image2field} \cdot \mathbf{X}_{field} \tag{3.4}$$

And the transform matrix $\mathbf{T}_{image2field} \in \Re^{3\times4}$ could be represented as three parts:

$$\mathbf{T}_{image2field} = \mathbf{K} \cdot \mathbf{T}_{camera2robot} \cdot \mathbf{T}_{robot2field} \tag{3.5}$$

In (3.5) what we can figure out from the corresponding point pairs is $\mathbf{T}_{image2field}$, and what we want to know is $\mathbf{T}_{robot2field}$. If we know the $\mathbf{T}_{robot2field}$ then we could directly read robot pose from this matrix. Also, the camera calibration matrix $\mathbf{K}$ is also already known because the calibration has been done before each match. $\mathbf{T}_{robot2field}$ can also be represented as (3.6):

$$\mathbf{T}_{robot2field} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} = \begin{bmatrix} cos\alpha & sin\alpha & 0 & T_x \\ -sin\alpha & cos\alpha & 0 & T_y \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.6}$$

There are 4 unknown parameters in (3.6), so at least 2 pairs of corresponding points are required to solve the equation. The 2 pairs of corresponding points can be selected by two goal posts or the start and end point of a specific line. Suppose $\mathbf{M} := \mathbf{K} \cdot \mathbf{T}_{camera2robot}$, elements in $\mathbf{M}$ are all known values, so the equation can be written as (3.7):

$$\mathbf{x} = \mathbf{M} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \cdot \mathbf{X} = \mathbf{A} \cdot \mathbf{X} \tag{3.7}$$

$$[\mathbf{x_1} \quad \mathbf{x_2}] = \mathbf{A} \cdot [\mathbf{X_2} \quad \mathbf{X_2}] \tag{3.8}$$

So, in (3.8) it shows the how did two pairs of corresponding points calculate matrix $\mathbf{A}$ which contains four unknown parameters. If we have more than 4 pairs of corresponding points, SVD method for minimum solution finding could be applied.

Chapter 4

Conclusion

4.1 Summary of achievements

This semester in the course "Introduction Lab Humanoid RoboCup" we worked based on open source framework codes and did some researches and improvements on both the kicking method and self-localization issues. The following will show you the achievements:

4.1.1 Kicking methods

1. Advanced kicking methods with different strength in different task situations
2. Kicking with both feet based on the distance of a robot and the ball
3. Improved alignment to the goal and area-based alignment for a better scoring performance

4.1.2 Self-localization

1. More robust penalty mark and line perception
2. Eliminate pose estimation approximation error

4.2 Future work

4.2.1 Introduce strategies for passing and dribbling

The kicking method based on different area could be combined with passing ball and cooperation with another teammates. Then the scoring performance will be improved quite a lot. Dribbling is the motion for kicking the ball for small steps for a long distance. This could avoid opponents getting the ball.

4.2.2 Implementation more methods of line perception

Although the robot now is able to detect the lines with large slope, but the very vertical line is still a problem. The more lines being detected on the field does not harm the self-localization result. But if the robot looks at the direction at the goal, the goal net gives rise to a big problem. In the experiment, since they were all detected as "lines", sometimes our goal keeper tried to go into the goal.
In order to enhance the robustness of our self-localization, we can consider about the other features or making the color calibration more stable. Now, if the illumination conditions change during the

game, the performance of our robot can not be guaranteed.

Based on the three points above, some potential solutions can be come up with as following:

- Adding the horizontal scan grids into our current scan grid. The trade off between computing time and the improvement of performance should be taken into consideration.
- Improving the goal detection as an important feature on the field. Compared with the feature of lines, the features of goal are more affine invariant.
- Using a special filter when the robot detects the goal to get rid of the influence by the goal net.

4.2.3 Come up with a more adequate noise model

This semester we focused on the visual odometry part, just the sensor measurement of the both two filter method. The motion model also plays an important role on the localization and is restricted with noise comes from actuators of each joint as well as the ground friction at the same time. So in the future, this part of work should be done and set up a more proper and scientific noise model to describe the uncertainty of motion model.

4.2.4 Search for ball

There is still some problem of searching for the ball, this part belongs not to the localization problem, but a tracking problem.

4.2.5 Data association error

According to the both current method and PnP pose estimation method, the correctness of data association determines the final quality of self-localization directly. Whether the current data association robust enough and which noise comes from, these issues should be more discussed.

4.2.6 Communication with another robots

This semester we only operate on each single robot without communication with another team players. The communication is very useful and it can gather more information and more fast. For example, if one of the robot has seen the ball, it could 'tell' other robot where the ball is. Then the other robot will not implement the ball search task any more and do another more efficient job. Furthermore, if two robot detect the ball at the same time, they could make a data fusion through communication to improve the accuracy.

Appendix A

Appendix

The following sections is part of the quick guide we received from our teaching assistant Zhiliang Wu.

In this chapter, a brief guidance is discussed, which records a quick and easy start for working with NAO. The purpose is that the future students are able to set up the whole environment in their personal PCs and debug everything on it with the help of this note, rather than only using the given Laptop provided by the Lab. The fundamental problems in calibration procedure are also discussed here to give useful suggestions, so that the future student can avoid the problems we met and spent a long time on. The version of code release is from B-Human RoboSoccer team in [16].

A.1 Building the project

After downloading the *BHumanCodeRelease-master.zip* from the git-repository of B-Human team, some more packages are required before building the whole project. For Ubuntu user, the first step can be done by implementing the command as:

```
sudo apt-get install qt4-dev-tools libglew-dev libxml2-dev clang graphviz
```

In addition, 32 bit version of *Naoqi* should be downloaded (more details can be found in [12]), and be installed with the command:

```
cd <BHuman-CodeRelease>/Install
./installAlcommon.sh <pathto32bitC++NAOQi.zip>
```

Then the whole project can be built with:

```
cd <BHuman-CodeRelease>/Make/Linux
make all
```

A.2 Flashing the robots

The step-by-step instruction can be found in [13](cf. 2.4). Here are some key-steps:

1. Use a USB with the required tools describe in [13].'openNAO-atom-system-image-2.1.4.13.opn' should be downloaded from the "Aldebaran"-website.

2. Connect the robot with network cable.

3. Plug in the USB drive into backside of the NAO and press the chest button for a few seconds until the chest flashes blue. The installation will start and the process will be monitored with the help of the LED lights around the robot's ears.

4. Press the chest button and write down the robot's IP-address, which is important for the further work.

A.3 Calibration

Calibration is very important for the robot. Each time before the game, the calibration should be implemented according to the field and the luminance environment. The calibration process can be done in the application *SimRobot*. A step-by-step guidance in [16](cf. 2.8) can be referred.

A.3.1 Joint Calibration

In order to ensure a stable performance of NAO, joint calibration should be finished first. During the semester, we adjust the joint calibration using the *JointCalibrator*. One thing to mention is that the changes of the calibration is only stored in *BHuman-CodeRelease/Config/Robots/Default* as *jointCalibration.cfg* with the command

```
save representation: JointCalibration
```

in *SimRobot*. To save it on NAO, the *copyfile* command should be implemented as:

```
cd <BHuman-CodeRelease/Make/Linux>
./copyfiles <Develop/Debug/Release> <IP_AAA.BBB.CCC.DDD>
```

A.3.2 Camera and Color Calibration

There are three modes in Camera Calibration, namely automatic, default and manual. The automatic mode will finish the calibration automatically, but with large deviation to the groundtruth. So the default and manual Calibration are preferred. They can be implemented in the console in SimRobot by

```
call CameraCalibrators/Default
```

The default calibration will calculate the result for the upper and lower camera according to the point chosen by us. What important for the point collection is that the point should be collected both in upper camera and in lower camera, so that it will converge faster and easier.
If the Default Calibration is not satisfying, the manual calibration can be used with

```
1 call CameraCalibrators/Manual
```

to finely tune the parameter. Figure A.1 shows an example of the camera Calibration.

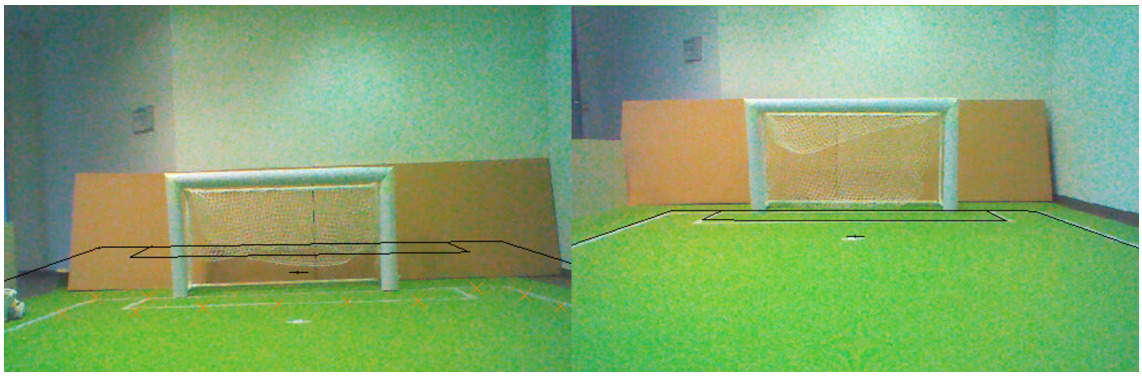


Figure A.1: Calibration: Before vs After

Color calibration is not as complicated as Joint and Camera Calibration. The only thing to take care is that the color of the ball is **orange**, rather than red.

A.4 Coding and Compiling

A.4.1 Tools for coding

To read and understand the framework of B-Human team in a easier and better way, *CodeLite* proves to be a great tool. B-Human provided a script to automatically generate the workspace for it., which can be found in *Make/LinuxCodeLite/generate*. After installing the *CodeLite*, the whole workspace can be opened like Figure A.2.
Different variables from different classes defined in various folders can be searched easily with the function "search in workspace" in *CodeLite*, which helps the understanding process from beginning.

A.4.2 Compiling and Testing in SimRobot

It is always exciting to test some new ideas immediately on the real robot. But before doing that, implementing the modified code by simulation can be of great help as in SimRobot everything is assumed to behave itself perfectly, such as the camera calibration and joint calibration of the robot. So it should at least behave as one wishes in the SimRobot, then the code can be moved onto the NAO to see the real effects. To test the modified function in simulation, the application *SimRobot* should be recompiled. From the experience of trial and error, the following command can be executed:

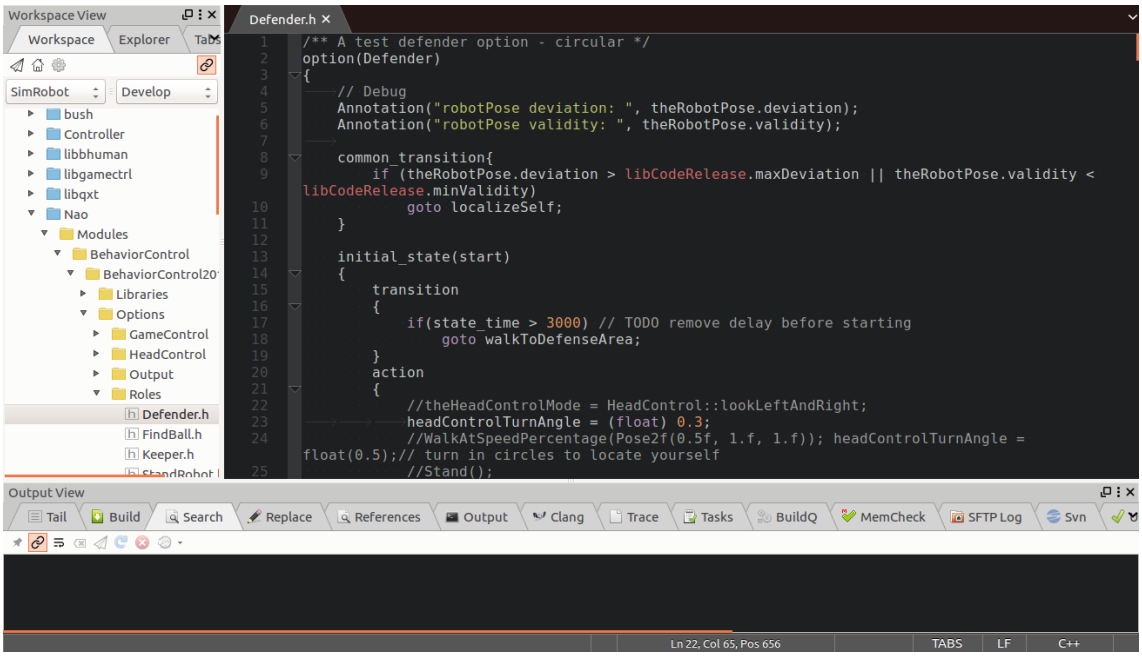


Figure A.2: CodeLite-Interface

```
cd <BHuman-CodeRelease>/Make/Linux
make all
```

As the project is already compiled before, this command will only compile the files which has been changed. So everything related will be compiled without too much consideration. To see the result in Simulation, run the following command:

```
cd <BHuman-CodeRelease/Build/Linux/SimRobot/Develop>
./SimRobot
```

The result can be shown by opening "Game2015Fast.ros" located in *BHuman-CodeRelease/Config/Scenes* with the application *SimRobot*.

A.4.3 Compiling and Testing on NAO

To change the code stored in NAO, the function "copyfiles" must be used. More concretely, the source file and configuration will then be copied into the 'head' of NAO, so that it can perform the new behaviors. Following command can be used:

```
cd <BHuman-CodeRelease/Make/Linux>
./copyfiles <Develop/Debug/Release> <IP_AAA.BBB.CCC.DDD>
```

where the first parameter is usually chosen as "Develop" and the second parameter is the Network-Address of the NAO you want to program, such as *10.0.29.171*.
What to mention here is that the configuration file is also transferred to the NAO. But only the configuration file located in *BHuman-CodeRelease/Config/Robots/Default*. So before *copyfiles*, make sure that the configuration are corresponding to the right one. Our method is to save the configuration file individually and replace it with the corresponding one if the robot changes.

A.5 Partial code of a .kmc file

```
  kpx = 0.3 ;
2 kix = 0.04 ;
  kdx = 0 ;
4 kpy = 0.3 ;
  kiy = 0.04 ;
6 kdy = 0 ;

8 {
  duration = 2000 ;
10 leftFootTra1 = {x = 0 ; y = 55 ; z = -230 ;};
  leftFootTra2 = {x = 0 ; y = 55 ; z = -230 ;};
12 leftFootRot1 = {x = 0 ; y = 0 ; z = 0 ;};
  leftFootRot2 = {x = 0 ; y = 0 ; z = 0 ;};
14 rightFootTra1 = {x = 0 ; y = -55 ; z = -230 ;};
  rightFootTra2 = {x = 0 ; y = -55 ; z = -230 ;};
16 rightFootRot1 = {x = 0 ; y = 0 ; z = 0 ;};
  rightFootRot2 = {x = 0 ; y = 0 ; z = 0 ;};
18 leftArmTra1 = {x = 0 ; y = 120 ; z = 80 ;};
  leftArmTra2 = {x = 0 ; y = 120 ; z = 80 ;};
20 leftHandRot1 = {x = 0 ; y = 1 ; z = 0.5 ;};
  leftHandRot2 = {x = 0 ; y = 1 ; z = 0.5 ;};
22 rightArmTra1 = {x = 0 ; y = -120 ; z = 80 ;};
  rightArmTra2 = {x = 0 ; y = -120 ; z = 80 ;};
24 rightHandRot1 = {x = 0 ; y = 1 ; z = -0.5 ;};
  rightHandRot2 = {x = 0 ; y = 1 ; z = -0.5 ;};
26 comTra1 = {x = 10 ; y = 0 ;};
  comTra2 = {x = 10 ; y = 0 ;};
28 headTra1 = {x = -2 ; y = -0.5 ;};
  headTra2 = {x = -2 ; y = -0.5 ;};
30 odometryOffset = {x = 0 ; y = 0 ; z = 0 ;};
  },
```

Acknowledgments

We would like to express our gratitude to:

- Prof. Dr. Gordon Cheng and Chair of Cognitive Systems for offering this course and all the hardware equipments.
- Our supervisor Mohsen Kaboli for his patient help and many constructive suggestions. He was always willing to share his experiences and gave us significant guiding. His office was always open to us. He was strict but we really learned a lot from that. Not only the experiment itself also on how to deliver a presentation.
- Our teaching assistant Zhiliang Wu, who prepared all the lab stuff for us and wrote detailed tutorials that let us know how to get familiar with the NAOs step by step. His careful work helped us to get started as quickly as possible and gave us a lot confidence to finish the task.
- Our whole team yellow members, Minkai, Fabian, Jingjie, Tianming, Zhiyi and Yao for the team spirit and the fun we had during so Mondays, Fridays and even weekends.

List of Figures

Bibliography

[1] Aldebaran. Joints nao v5 and v4. http://doc.aldebaran.com/2-1/family/robots/joints_robot.html.

[2] Tony F Chan, Gene H Golub, and Randall J LeVeque. Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247, 1983.

[3] RoboCup Technical Committee et al. Robocup standard platform league (nao) rule book, 2016. Only available online: www.tzi.de/spl/pub/Website/Downloads/Rules2016.pdf.

[4] Majid Lashgarian Mostafa Yaghobi Mohammad Shafiei R. N. Ehsan Hashemi, Maani Ghaffari Jadid. Particle filter based localization of the nao biped robots. 2012. 44th IEEE Southeastern Symposium on System Theory.

[5] Simon Flückiger. Self-localization on nao robots based on external features: Pnp pose estimation, 2016. Only available online: https://drive.google.com/file/d/0B8OEfzrd1JHkbkNqMkJYT0tZbUk/view.

[6] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2, 1999.

[7] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[8] AR Ismail, M RA Mansor, M FM Ali, S Jaafar, and M SNM Johar. Biomechanics analysis for right leg instep kick. *Journal of applied Sciences*, 10(13):1286–1292, 2010.

[9] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 2. Pearson Education, 1998.

[10] Dany Manickathu. Generating kicking motions for the nao, eth report. https://drive.google.com/file/d/0B_88sb4SLadNMWlZZTI1QmFIS2M/view, 2014.

[11] Filippo Martinoni. A framework for passing, 2014. https://drive.google.com/file/d/0B_tMocQbHqQuOUMteFpELUxMaHM/view.

[12] RoboTUM. Robosoccer wiki. https://gitlab.lrz.de/ga68hom/RoboCup/wikis/home.

[13] Thomas Röfer, Tim Laue, Judith Müller, Michel Bartsch, Malte Jonas Batram, Arne Böckmann, Martin Böschen, Martin Kroker, Florian Maaß, Thomas Münder, Marcel Steinbeck, Andreas Stolpmann, Simon Taddiken, Alexis Tsogias, and Felix Wenk. B-human team report and code release 2013, 2013. Only available online: http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf.

[14] Thomas Röfer, Tim Laue, Judith Müller, Michel Bartsch, Malte Jonas Batram, Arne Böckmann, Nico Lehmann, Florian Maaß, Thomas Münder, Marcel Steinbeck, Andreas Stolpmann, Simon Taddiken, Robin Wieschendorf, and Danny Zitzmann. B-human team report and code release 2012, 2012. Only available online: `http://www.b-human.de/wp-content/uploads/2012/11/CodeRelease2012.pdf`.

[15] Thomas Röfer, Tim Laue, Judith Müller, Armin Burchardt, Erik Damrose, Alexander Fabisch, Fynn Feldpausch, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, Daniel Honsel, Philipp Kastner, Tobias Kastner, Benjamin Markowsky, Michael Mester, Jonas Peter, Ole Jan Lars Riemann, Martin Ring, Wiebke Sauerland, André Schreck, Ingo Sieverdingbeck, Felix Wenk, and Jan-Hendrik Worch. B-human team report and code release 2010, 2010. Only available online: `http://www.b-human.de/downloads/bhuman10_coderelease.pdf`.

[16] Thomas Röfer, Tim Laue, Jesse Richter-Klug, Maik Schünemann, Jonas Stiensmeier, Andreas Stolpmann, Alexander Stöwing, and Felix Thielke. B-Human team report and code release 2015, 2015. Only available online: `http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf`.

[17] Thomas Röfer, Tim Laue, and Dirk Thomas. Particle-filter-based self-localization using landmarks and directed lines. pages 608–615. Springer, 2005.

[18] Raphael Stadler. Robocup: Ball dribbling, 2015. `https://drive.google.com/file/d/0B_tMocQbHqQuSnFubFlveDM5cnM/view`.

[19] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. 2005.