System Design Project Final Report

Group12 - School of Informatics

Seyed Behzad Tabibian

I. Introduction

II. Robot Construction

Since our earliest robot designs, our design philosophy was to have a robust, lightweight and fast robot, which would be able to outmanoeuvre opponents on the pitch whilst being robust enough to withstand collisions with bigger opponents and walls.

Reviewing the archives of previous SDP years, as well as closely following the developing designs of the other groups in our year, showed that this simple strategy was one that had been rejected, or possibly overlooked, by almost all other groups over time. Instead, designs that were favoured featured box-like structures with a lot of extra weight and a lack of agility. Due to the success of the previous years winners, holonomic wheels were considered, but were rejected due to not providing sufficient advantages to outweigh the extra effort necessary in their implementation, especially when this effort could be focused into other equally important areas such as strategy and vision, which we believed would be important in the final games. We also believed that a well designed wheel base, using non-holonomic differential drive wheels would allow for an equally manoeuvrable robot if used with an effective movement algorithm.

Researching into gears discovered a 3 - 10 ratio would make the robot up to three times faster than if the wheels were connected directly to the motors[1]. Once gears were incorporated on the robot, whilst improving the speed as predicted, they did affect the straight line ability of the robot, as shown below: Clearly increased speeds led to massive offsets in travelling in a straight line. After testing multiple different options, including changing caster wheels, using ball-bearings, and putting the motors at the front, back and centre of the robot, it was eventually realised that the robot would diverge from its path whilst moving forwards, but would travel in a relatively straight line whilst reversing (at the same speed). Therefore a simple solution was to switch the direction which the motors faced (i.e. motors reversed $\rightarrow$ robot moved forward, and vice-versa). This led to much improved results:

TABLE I
Distance off-line (Running for 10seconds)

Speed (rpm)	Off-Centre (mm)
1.5	82
3	216
4.5	850
6	N/A - Span in Circles

TABLE II
Distance off-line 2 (Running for 10seconds)

Speed (rpm)	Off-Centre (mm)
1.5	0
3	4
4.5	10
6	46

During the early stages of the build process we did not take into account that we were designing an actively controlled system, where robots path is being altered with every vision frame processed. Therefore these reduced discrepancies could be managed by the programming of the movement. By the end of its development, despite its small, slight appearance, our robot was consistently one of the sturdier designs in the competition, only once losing a single piece from its structure due to a collision on the pitch. Our fast, agile design also proved to be effective alongside our robust strategy, as we went on to achieve a semi-final position in all the friendly tournaments, before unfortunately being knocked out in the quarter finals on the final day. We were also among the teams with the highest number of total goals scored across all of our played matches, 16 in 8 matches!

III. Strategy

The main focus was to create an attacking strategy which would utilise the speed of the robot. The strategy was split into two sections, the high level

[1]http://www.ecst.csuchico.edu/%7Ejuliano/csci224/Slides/03%20-%20Gears%20Pulleys%20Wheels%20Tires.pdf

areas (where the robot will move to, when to kick, dribble etc), and the lower level areas (coding the movement of the robot).

A. High Level Strategy

Initially we created a simple state system strategy, which checked the current on-pitch situation (using the data sent from the camera) and decided the appropriate strategy to run. The state system approach was used because new states could be added easily, allowing for multiple strategies. Our strategy was based on defining a point to move towards. A point class was created, holding the x,y co-ordinates of points on the pitch. This class was extended to create instances of the robot, ball, goal, and any other necessary points for use in the strategy.

An optimum point was implemented (Algorithm 1), which would be a defined distance behind the ball at the same angle as the ball to goal angle, such that the robot would move to this point, and then to the ball, to ensure the robot was facing the goal when it reached the ball. This worked well in practice, working everytime when the robot was behind the ball, and 9/10 times when the robot was inbetween the goal and the ball (i.e. had to navigate around the ball).

$threshold \leftarrow 70$
$(x_1, y_1) \leftarrow ball(x, y)$
$(x_2, y_2) \leftarrow goal(x, y)$
$ballGoalAngle = atan2((y_2 - y_1), (x_2 - x_1))$
$xOffset = threshold(\cos(ballGoallAngle)$
$yOffset = threshold(\sin(ballGoalAngle))$
$(x_3, y_3) = (x_1 - xOffset, y_1 - yOffset)$
return (x_3, y_3)

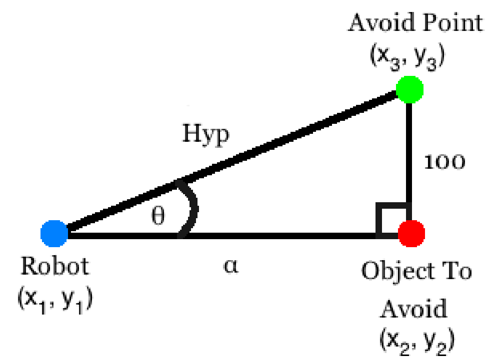
Algorithm 1: Caclulate Optimum Point

This function was later abstracted to a higher level, modifying the function to take any two points and return the angle between them, to increase its range and decrease repetition of similar code.

The next target was to be able to navigate around objects. Points were used again, utilised in a function which would calculate an avoidance point at a $90°$ angle and a defined distance from the object to avoid. The function would return the avoidance point, and the robot would move to it.

This function was dynamic, and the point would move as the robot moved, allowing the robot to navigate the object more smoothly.

The function calculated the avoid point using trigonometry, as shown in the following diagram and pseudo-code:



$threshold \leftarrow 100$
$(x_1, y_1) \leftarrow Robot(x, y)$
$(x_2, y_2) \leftarrow ObjectToAvoid(x, y)$
$\alpha = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
$hyp = \sqrt{\alpha^2 + threshold^2}$
$\theta = \sin(\frac{100}{hyp})$
$xOffset = hyp(\cos\theta)$
$yOffset = hyp(\sin\theta)$
$(x_3, y_3) = (x_1 + xOffset, y_1 + yOffset)$
return (x_3, y_3)

Algorithm 2: Caclulate Avoid Point

Again, this function was abstracted to a higher level to calculate the avoidance for any points, as the robot may need to avoid a ball or a robot, which was useful for the function created to avoid the ball when moving the robot to get to the correct side of the ball (so the ball would be between the robot and the goal we were attacking). Alongside these additions, functions were implemented which decided if the ball was in more difficult positions, such as being close to the walls or in the corners, for use in the state system, so different strategies could be brought in to deal with these situations. The strategies were further adapted to readjust to a change in the goal we were attacking.

A more precise shooting system (i.e. being able to shoot directly into the corners of the goal) and a defensive strategy would have improved our performance in the final competition, as a number of shots were saved in the centre of the goal, and we had no real strategy to deal with the opponent having possession, and whilst usually the attacking strategy would cover most defensive situations, we were found out once or twice in the final game. Other than these faults, the strategies worked successfully, and made good use of the robots speed.

B. Low Level Strategy

In low level motion planning layer of the agent architecture we needed a robust algorithm capable of compensating possible noise incurred by output of vision stack. The simple goal for this part is to make robot capable of following a path to reach a destination position determined by higher levels of path planning algorithm. A. Potential Field To achieve the goal stated above we started with implementing Potential Fields[2] algorithm. This algorithm is a well-known method for robot path planning. It has several properties which makes it suitable for our application. The behaviour of the algorithm is completely reactive and therefore if it is fed with noisy input in one cycle it will be able to recover in next cycles, therefore performance of the algorithm degrades as amount of noise increases never failing completely. C. Challenges During implementation of this method, we had several challenges. We started testing this algorithm for milestone two but failed. The initial intuition for most members was that the failure was due to complexity of this algorithm while the failure was in fact due to unreliable output produced by vision stack. Later, new set of tests showed a very good performance once the vision stack was robust enough to provide reliable input for the algorithm. Successful implementation of this algorithm become a key to our success in third and fourth friendly matches. D. Kinematic Model and Implementation In order to implement this algorithm successfully we needed to find a way to apply the calculated velocity vector on the robot. To do this, we used a kinematic model of a differential drive robot. In figure FIGURES ,the velocity vector is decomposed into linear and angular velocity and later they are fused to calculate velocity of left and right wheels. Integrating all this we were able to implement a successful motion planning algorithm which distinguished our team from other teams. E. Results A sample result of running this algorithm on the pitch is demonstrated in figure FIGURES, Overall, this algorithm could deal with almost all scenarios successfully and when integrated with a high level decision making layer, we had a reliable game play scenario. The solution also can deal with obstacles effectively but this feature was not used because obstacle avoidance was dealt with by higher levels of decision making. Therefore, the algorithm at this level is a conventional P controller. F. Lessons Learned It is possible to implement reliable applications based on ideas developed in research labs. Failure of an algorithm at higher levels sometimes may be due to invalid inputs from lower layers. Simulation at abstract level is very useful for making sure that an algorithm is developed according to specification but successful implementation based on such simulation environments does does not guarantee optimal performance in real environments.

IV. PERCEPTION

The robot was able to perceive the environment it is situated in using the overhead camera. Using the images provided by the camera, we were able to calculate robot and opponent location and orientation, ball location on the pitch and the pitch itself.

A. Initial approach

Finding locations of different objects proved to be easy using simple methods like colour segmentation. The main challenge was in calculating robot orientation. Initially we used the dark spot and the T on the plate. This method was not very successful, since isolating the dark spot proved to be hard and inaccurate.

B. Machine Learning

Later a solution based on Machine-Learning methods was developed. We implemented solutions using SVM and Bayse algorithms available in the OpenCV library. A range of features such as hue moments, compactness and area were used for training a model for each object. We also used distance between two objects, by using this feature we were able to first find a principal object T and then using the distance feature find the correct dark spot close to this object.

C. Central Image Moments

Even though the Machine-Learning based solution produced reliable outputs but we realised that for a problem of this size, there should be simpler methods available. After studying the literature of computer vision, we implemented a method using second order of central image moments. This method is based on calculating main inertial axes, around which the object can be rotated with minimal or maximal inertia. For detailed description of this method reader is referred to[**?**]. Using this method orientation of an object is computed as follows:

$$\mu'_{20} = \mu_{20}/\mu_{00} = M_{20}/M_{00} - \bar{x}^2 \tag{1}$$
$$\mu'_{02} = \mu_{02}/\mu_{00} = M_{02}/M_{00} - \bar{y}^2 \tag{2}$$
$$\mu'_{11} = \mu_{11}/\mu_{00} = M_{11}/M_{00} - \bar{x}\bar{y} \tag{3}$$
$$\Theta = \frac{1}{2}\arctan 2\left(2\mu'_{11}, \mu'_{20} - \mu'_{02}\right) \tag{4}$$

One can observe that equation 4 produces results from $-\pi/2$ - $\pi/2$. Our method cannot compute the heading of the calculated vector, to disambiguate this issue we used another method that was developed in earlier stages. It uses central moments of the object and the circle fitted around the object to calculate the orientation2. This combined solution could produce reliable output. Even though our method calculating robot orientation was not very robust against noise at the specified areas, it was very accurate in 85 to 90% of the cases.

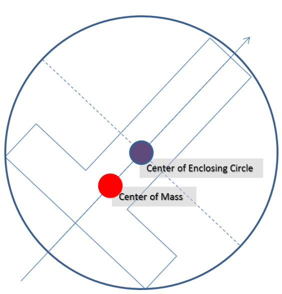


Fig. 1. Calculating orientation using center of mass of object and surrounding circle, used as a helper for second order of central moments

D. Results

Using second order of image moments for calculating orientation of objects was effective for this project. Implementing this method was one of the keys to our success in milestone 3(score 70) and first friendly match(becoming semi-finalist) and later in other matches. Our initial implementations had a very poor performance of about 5fps, however by the end of the project and using simpler methods we reached frame rates of over 22fps. Our program however suffered from major drops in frame rates on some specific machines in the lab, we could not successfully resolve the issue since the problem was not easily reproducible. This issue was one the major reasons we could not perform very well in the final match. A few other achievements in this part of the project was to learn how to effectively use OpenCV library in C. We also learned about Bayes and SVM classifiers and compare their performance. The developed library contains:

- Various methods of object isolation methods such as background subtraction and colour segmentation. Various methods of calculating orientation as described earlier.
- Complete implementation of Machine Learning methods for object recognition.
- Utility programs for tuning the colour segmentation thresholds.
- Utility methods for changing functionality of program based on input parameters.

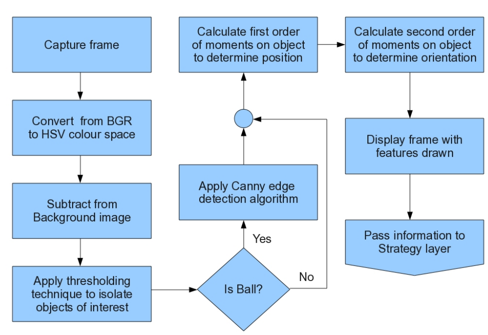


Fig. 2. Final flow chart of vision program

V. LOW LEVEL MOTION

In low level motion planning layer of the agent architecture we needed a robust algorithm capable of compensating possible noise incurred by output of vision stack. The simple goal for this part is to make robot capable of following a path to reach a destination position determined by higher levels of path planning algorithm.

A. Potential Field

To achieve the goal stated above we started with implementing Potential Fields[?] algorithm. This algorithm is a well-known method for robot path planning. It has several properties which makes it suitable for our application. The behaviour of the algorithm is completely reactive and therefore if it is fed with noisy input in one cycle it will be able to recover in next cycles, therefore performance of the algorithm degrades as amount of noise increases never failing completely.

B. Challenges

During implementation of this method, we had several challenges. We started testing this algorithm for milestone two but failed. The initial intuition for most members was that the failure was due to complexity of this algorithm while the failure was in fact due to unreliable input produced by vision stack. Later, new set of tests showed a very good performance once the vision stack become robust and reliable. Successful implementation of this algorithm was a key to our success in third and fourth friendly matches.

C. Kinematic Model and Implementation

In order to implement this algorithm successfully we needed to find a way to apply the calculated velocity vector from previous step on the robot. To do this, we used a kinematic model of a differential drive robot. In figure **??**,the velocity vector is decomposed into linear and angular velocity and later they are fused to calculate velocity of left and right wheels.

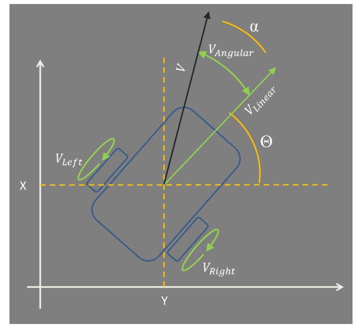


Fig. 3. Kinematic Model of Non-holonomic Differential Drive Robot

$$V = k_{att}.(Pos_{current} - Pos_{destination}) \quad (5)$$

$$V_{linear} = \left\| v \right\| cos(\theta), V_{angular} = \frac{K\theta}{\pi} \quad (6)$$

$$V_{left} = V_{Linear} - rsin(V_{Angular}) \quad (7)$$

$$V_{right} = V_{Linear} + rsin(V_{Angular}) \quad (8)$$

Integrating all this we were able to implement a successful motion planning algorithm which distinguished our team from other teams.

D. Results and Lessons Learned

A sample result of running this algorithm on the pitch is demonstrated in figure 4, Overall, this algorithm could deal with almost all scenarios successfully and when integrated with a high level decision making layer, we had a reliable game play scenario. The solution also can also deal effectively with obstacles using Extended Potential Field algorithm[?] but this feature was not used because obstacle avoidance was dealt with by higher levels of decision making. Therefore, the algorithm at this level is a conventional P controller.

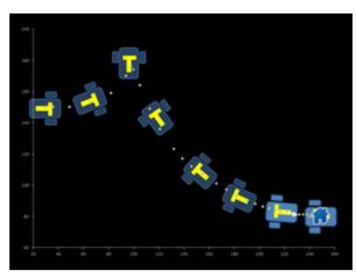


Fig. 4. Complete Robot movements from start position to destination position.

E. Lessons Learned

- It is possible to implement reliable applications based on ideas developed in research labs.
- Failure of an algorithm at higher levels sometimes may be due to invalid inputs from lower layers.
- Simulation at abstract level is very useful for making sure that an algorithm is developed according to specification but successful implementation based on such simulation environments does does not guarantee optimal performance in real environments.