Beaver Works Summer Program Technical Report

Michelle Tan

***Abstract*—This material is part of a summer boot camp at MIT Beaver Works where teams of high schoolers program cars to compete in the Mini Grand Prix Challenge, an autonomous car race around a track. This four-week program meets on weekdays for eight hours each day. The program is based off the 6.141 Robotics Science and Systems class and the Robotics IAP at MIT. Each day typically consists of three lab periods, a technical lecture, a seminar with distinguished guest speakers, and an occasional communications class.**

I. Introduction

THE purpose of this program is to begin to understand the algorithms and methods used today in autonomous navigation and to experiment with them on a 1/10-scale racecar. Teams explore the abilities and limitations of building robust yet fast robotics systems through tasks such as following a wall, detecting a colored blob, following a colored blob, avoiding obstacles, using a color to decide whether to take a turn, and optionally implementing localization and mapping. Much of the class dealt with problems inherent to autonomous navigation, like sensor imperfections, accurate object detection from camera images, and speed constraints.

The cars used were referred to as RACECAR, which stands for Rapid Autonomous Complex-Environment Competing Ackermann-steering Robot. The cars were pre-built with its components prior to the start of the program, which include the NVIDIA quad-core CPU, a 192-core GPU, an Ackermann-steering system, and a large variety of sensors including a lidar sensor, camera, and odometer. The teams used ROS, a ”meta” operating system for robots that focuses on modularity, interfaces, and code reuse. [1]

II. Week 1

A. Technical Goal

In the first week, the learning objective was to get experience in working with the sensors and actuation of the robot and to also learn about control systems. The end-of-week challenge consisted of being able to follow a left and right wall at a desired distanced away independent from the starting position from the wall.

B. Approach

The teams were required to use lidar to sense the wall but they were free to implement a state controller of their choice.

1) Bang Bang Controller: The simplest of the controllers is using a Bang Bang controller, which states that if the robot is too close, it should turn away and if too far, turn towards the wall.

2) PID Controller: A more sophisticated method is to implement a PID controller which does calculations on the error value to determine an optimal steering angle that creates a smooth oscillation to correct for the robots distance. There are three parts to the PID controller: Proportional, Integral, and Derivative. They are incorporated in the equation below where u is the controller output and e is the error

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d e'(t) \tag{1}$$

The proportional part allows the steering angle to scale linearly based on the error. The integral portion scales linearly with the total error over time. The derivative portion scales based on the derivative of the error in order to limit the overshoot. [2]

3) Two Point Controller: When calculating the error distance, there was an additional consideration of which laser scan data to use. The angle of the robot could easily throw off the reading if only one measurement were taken, since the distance measured perpendicular from the robot won't necessarily be the true distance as shown below.

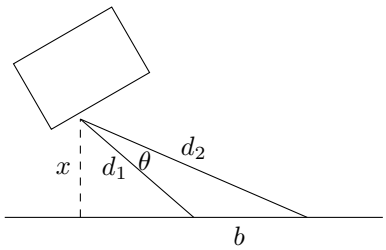

Fig. 1: Diagram of angled robot

A two point controller could account for this by using basic trigonometry to determine the true distance from the wall. Given two laser measurements d_1 and d_2 with a known θ between them, the following formula can calculate the true distance x between the wall and the robot.

$$x = \frac{d_1 d_2 sin(\theta)}{\sqrt{d_1^2 + d_2^2 - 2d_1 d_2 cos(\theta)}} \tag{2}$$

Since there were so many options and there wasnt enough time to test all of them, we wrote code implementation of all of the separate features, like the PID controller and error calculation, so we could test each systematically to find the best option. We also decided to approach the challenge by starting simple and incrementally improving it.

C. Process

To calculate error, we decided to find the true distance to the wall by finding the minimum point in a range of

laser distances, since mathematically, the true perpendicular distance is the shortest distance. This assumes that the robot is rotated such that the minimum distance is in the range of laser values taken. Given the parameters of the wall follow challenge, we thought this was a safe assumption. The points taken from the laser were in the range starting from 2.5°behind the robot to 5°ahead of the robot.

After we could find the error, we started with the bang bang controller. This simple implementation was able to reliably follow a wall, even if the wall was curved. The problem with the Bang Bang control was when we started the robot far away from the line. Since the turning rate was independent from the actual distance from the wall, when the robot was far away, it would turn at a slower rate than it intuitively should given the distance away. Even worse was that when the robot was close to the line, it would continue to use too steep of a steering angle, resulting in perpetual oscillations and jerkiness.

We next tried implementing a P controller in order to control the steering angle more. In a P controller, the error is proportionally scaled to be the steering angle, so bigger error would result in more severe turning and on the other hand, smaller error would result in a more subtle turn. Higher k_p values, would result in more responsiveness but also more unnecessary oscillations. We found that k_p values near 1.0 achieved the best balance of reactiveness but not overreaction. The P controller was significantly better at correcting for large errors, but at very high distances, it would sometimes turn back so quickly that it would smash into the wall.

Afterwards, we tried moving to a PD controller. The Derivative portion would calculate the rate of change of the error and limit the change in order to attempt to smooth out the oscillations. After a lot of tuning of both constants, we found that the derivative caused jerkiness no matter what value we had it at. This was most likely the result of noise. Although it was jerky we also found that it was necessary for adjusting the distance when the robot was really far.

In the end, a large majority of the challenge had to do with finding the right constants. We weren't able to find working constants for the entire PID controller due to lack of time, so we used a PD controller for the race. The setup that ended up working well was with a k_p of 1, a k_d of .05, and a k_i of 0. [PID table] With more time, we could have continued perfecting the numbers, but they were sufficient for this challenge.

D. Results

The challenge consisted of a straight left and right wall that the robot should be able to follow. Three time trials were recorded on the different sides starting the robot at different distances, then a challenge test was tried where the robot was put in the center of the 2-lane track, requiring agile recovery to avoid crashing into the wall. Every group was able to follow the wall, but few teams were able to correct the distance from the center of the track.

We learned that although there is a lot of theoretical research on optimal values for the PID controller, what happens in real life is vastly different. Amount of time needed for testing is significantly higher than the time to right the code. We found that what should happen or what happens in simulation is not a substitute for real life testing. Our car was able to complete the race with the following scores:

TABLE I: Final Challenge times

Left Wall	8.77
Right Wall	8.85

Although we were 8th out of 9 teams, the times were all within .2 seconds of each other so external factors from our algorithm such as battery level or timer inaccuracy could also have an effect. With more time, the constants could have been more tuned to a more optimum value and also included a k_i value.

III. Week 2

A. Technical goals

The challenge from this week was to use the ZED camera to sense a colored blob and use visual servoing to follow the blob. When the car was close enough, the robot had to make a correct turn based on the color of the wall where green indicated a right turn and red indicated a left turn. Afterwards, the program had to successfully wall follow until a certain point. The following diagram is taken from the lab handout for the week 2 challenge.

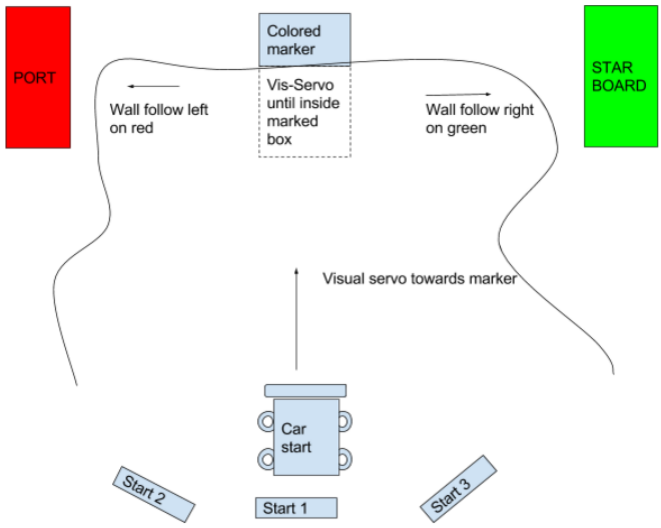

Fig. 2: Challenge diagram [3]

B. Approach

We started with getting a good blob detections algorithm. Then we came up with a simple but effective algorithm to have the robot drive towards the blob. It would then have to turn 90 °before transitioning into the wall following program from last week. We split up the task into a blob follow node and a turn node in order to facilitate testing and so we could split up the work effectively.

C. Process

We first focused on being able to detect a colored blob using the OpenCV library. In a lab session, we worked individually on each getting good blob detection. Everyone in the team put the working parts of their code together and we built off the final one. There were many possible approaches to find blobs such as contiguous colors and edge detection, but we decided to detect contours. According to OpenCV documentation, contours are "a curve joining all the continuous points (along the boundary), having same color or intensity." [4] We first use masks to filter for the desired color and changed the image to gray scale in order to make the contours more accurate. After drawing the contours, it would check if the size is larger than a certain threshold in order to eliminate noise. If the program determined that it was a blob, it would publish the detections to a separate node using a custom message.

While part of the team was working on publishing blob detections, others worked on visual servoing. The simplest algorithm was a bang bang method based on height and x position of the blob. If the blob was too short that meant it was too far so it would go forwards. If the blob was to tall, it meant that it was close enough and the robot would stop. If the x position of the blob was on the right side of the screen, the robot turned right to orient with the blob. Likewise, if the x position of the blob was on the left side, the robot turned left. As expected with a bang bang controller, it was jerky and when we were close to the target, it had trouble getting oriented because if it was a little off center, it would turn back too much and be stuck in an oscillation cycle. We decided to implement a PID controller into our visual servo, which helped with this issue.

We also had to modify our wall-follow code from last week and optimize it for our situation. We didnt focus on tuning this part too much because we knew it functioned from last week. Most of the work was integrating it to work with the other nodes.

This week, the structure of the task and how the nodes would communicate was essential to getting a working program. We had a blob detection node that would publish a custom blob message onto a visual servo node, which would then move towards the blob. When this node was finished, it had to send exactly one message to the wall follow node indicating whether to take a left or right turn. Eventually, we were able to successfully integrate them together. Due to miscommunication, we realized that no one had programmed the turn for when it got to the wall. We struggled to get that part working for the race because of lack of testing time.

D. Results

At the very last minute we were able to get the robot to turn and follow properly when it got to the wall. Unfortunately, it only worked on the green side because we figured out later that there was a bug in our code where the isGreen variable we were modifying was a local variable so it wasn't stored between functions. Then during the race, our robot didnt move probably due to lack of battery. After we charged it, the program did not work the same way and the gears had a glitch, so unfortunately we couldnt finish. Out of all of the teams, only 2 were able to finish. We learned important lessons in communicating to make sure everything is getting done and to make sure the robot is charging.

IV. Week 3

A. Technical goals

Our goal was to use reactive-based planning-based on what the robot can see locally in order to explore unbounded space while avoiding obstacles. We also had to detect blobs along the way and save the picture with the blob color and locations indicated. To add on from the image detection from last week, additional colors to detect were added(blue and yellow), and there was a challenge task to detect images of Ari, Sertac, a cat, and a robot.

B. Approach

Based on the open-ended nature of this weeks assignment, we had a team meeting where we discussed a lot of possible methods for obstacle avoidance and ways to take the shortcut. We first focused on ways to get around without obstacles. We had different people explore different ideas. In the end, we ended up using a potential field method that we learned about in a lecture, which was the most effective.

C. Process

Here are the methods we explored in the order we considered them

1) Right wall follow: If there are no obstacles and were not supposed to take the shortcut, we thought that a simple wall follow should be sufficient. However, we found that because of the sharp corner, it wouldnt be an option. As the robot approached the corner, it could never determine that it should turn until it had already run into the wall, since the robot looks out to the side and not directly in front of it. After we adjusted the field of vision to angled further up front, it was able to turn at the corner only at very low speeds.

2) Equal Distance wall following: Assuming there are no obstacles, an effective way to take the shortcut would be to simply follow the wall on both sides by trying to keep the distances on the left and the right the same. When the robot got to the shortcut, it would see a large spike in the laser reading for the left side and therefore turn left to account for it. However, it was determined that this model would break down with obstacles.

3) Obstacle detect: Look ahead and if obstacles are detected on the left side, take the right lane and vice versa. If both left lane and right lane have obstacles, take the center lane. The idea of changing the desired distance of the wall follow based on where the obstacles were was solid, but too many assumptions were made in the model

4) Left/Right Wall Follow: If there are no obstacles, follow the left wall the whole time, then switch to following the right wall when there is the shortcut to ensure that the robot takes the long way across.

5) State machine: We explored enough options for the case of wanting to take the shortcut and not taking the shortcut, along with how to avoid obstacles that based on where it is, that we could incorporate them all into a state machine. [Possible elaborate]

6) Open space: Use the scanner to find the largest contiguous open space and go there. This method could work well, especially for avoiding obstacles. We ran out of testing time to try it out, however.

7) Potential field: The potential field method uses the idea of charges attracting and repelling to control the robot. The robot is repelled from obstacles, but also repelled from behind so it continues to move. We ended up this method because it was the most robust. This implementation was the most effective and also the most simple at under 20 lines of code. [Diagram to explain]

For the blob detections, we used the version we made last week and added in the new colors. We came up with two possible solutions that we could have had working with more time. The first one was to average the HSV values of the pixels in each of the challenge images and hope that they are different enough that we could distinguish them. This would be messed up in different lightings, but since we arent penalized for incorrect labelling, we thought it could be worth it. [Show example] The second method would to do some sort of statistical analysis of the HSV values of the pixels of the original images. Then, we could find the distribution for the images from the ZED camera and compare them to find the closest match. This process was simplified because we found cv2 functions for calculating and comparing histograms. [Show examples] Since this method seemed more robust to lighting changes, we started implementing this method.

D. Results

In the final challenge, we used the potential field method. We ran out of time to execute the challenge images and we werent able to tune the color values well, so we missed a lot of colors in the actual challenge. We should have spent more time tuning the colors instead of focusing on the challenge images. We ended up coming in 3rd place with 36 points, which came from detecting four blobs and having two collisions.

V. Week 4

A. Technical goals

This week was dedicated to preparation for the Grand Prix Challenge race, but in addition there were two tech challenges that would be informally showcased.

1) Tech Challenge 1: In the first tech challenge, the robot had to publish blob detections while it explored an unstructured space with obstacles.

2) Tech Challenge 2: In the second tech challenge, the robot had to go along the racetrack and look for a colored blob that would be located at a forked intersection. A green blob would indicate an open shortcut that the robot could take while a red blob would indicate that the shortcut was blocked and the robot has to take a longer way around.

3) Grand Prix Race: The Grand Prix Race is the culmination of all of our work done thus far in the summer. The cars were supposed to race around the track, avoiding nearby cars. In addition, one point in the track there was a fork with two different paths: a straight shortcut and a longer path curving on the right side. Based on the color of a blob placed in the intersection, the car would have to determine if the shortcut was open where a red blob indicated a blocked shortcut.

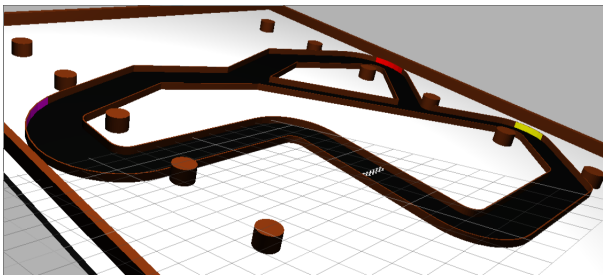

Fig. 3: Model of racetrack [5]

B. Approach

Since this Grand Prix was the only task that was scored, we wanted to focus on that task and make sure we had enough time to test it. In addition, the 2nd tech challenge was part of the Grand Prix challenge so it wouldn't require extra work if the the Grand Prix code worked. Any extra time we had would be spent working on the 1st tech challenge. Since we would be switching locations to the Walker Memorial which had different lighting and was also prone to further lighting changes due to natural light, we tried to speed up the process of fine-tuning the HSV values using a calibration program.

C. Process

1) Detecting blobs and exploring space: This challenge builds on the program we had last week for reactive planning, which was able to detect red, yellow, green, blue, and pink blobs. We wanted to continue testing and improving on the challenge images implementation from last week. In addition, the colored blobs could also be circles, rhombi, x shapes, and plus signs so the program had to distinguish between them. We found that a quick way to distinguish between them would be to inscribe each shape in a rectangle and compare the ratios of the area of the shape to the area of the inscribed rectangle. [Draw pictures and show ratios]

2) Navigating around the track: We decided to try modifying the reactive obstacle detection program from last week that used potential fields to guide the robot away from obstacles. This program was the most robust to being able to deal with all different kinds of obstacle configurations. The only modification that seemed to be needed was a larger imaginary charge behind the robot so it wasn't tempted to go backwards. In addition, we noticed that the track contained 4 major left turns and 2 major right turns(one of these being the intesection). When on a left turn, the most efficient path would be to stick to the left wall more, minimizing the distance. [Insert diagram]. Since all of the cars have the same maximum speed, we realized that we could get an edge by staying on the

inside lane. We implemented this by adding a boost constant to the force to the left when calculating the vector of least resistance. This required a lot of tuning to ensure that it would favor the left while not crashing into it.

3) Detecting shortcut availability: Instead of calibrating the robot to look for red or green, the robot looked for only red. If there was red detected, it would take the long way. Otherwise, it would assume that the shortcut is open. In order to make the right turn successfully we had to modify the forces on the robot to make it favor the right side. We had a problem with as the robot was turning, when it lost sight of the red blob it would stop turning. In addition, the left wall affinity would cause the robot to veer left so aggresively after turning that it would crash into the wall. We had to set up a counter that would execute as soon as it saw the red blob. For the first part of the counter, it would highly favor right turning until it had basically finished the turn. For the second part of the counter, it would switch lanes to the center to ease the transition to the left wall. Then, lastly the default track navigation code would continue as normal.

D. Results

1) Tech Challenge 1: Our robot was able to navigate while avoiding obstacles. It was able to detect colored blobs, but unfortunately we didn't put enough time into testing to have it reliably detect the challenge images or the alternate shapes.

2) Grand Prix Race: Our team did really well. We were ranked 1st out of all 9 teams in both the time trials with the following times:

TABLE II: Time Trial Results

Lap #	Color of blob	Time
1	Green	28.93
2	Red	33.46
3	Red	33.28

TABLE III: Final Grand Prix Results

Color	Time
Green	28.00
Red	38.20

VI. Conclusion

The goals in the program were to learn about robot motion, sensors, object detection, object avoidance, and SLAM(Simultaneous Localization and Mapping). We managed to explore all of these topics expect for Localization and Mapping. The Lidar's data wasn't detailed enough to be able to get a reasonably accurate map of the track with the obstacles. The results were an autonomous robot that could detect colored blobs, follow a wall, avoid obstacles, and follow a track.

I learned about ROS and how the structure of modular programming allows for greater complexity. I also learned about the different controllers that could be implemented and the advantages of closed loop control over open loop control. We learned about methods used for detecting colored object which included looking for contours and edges of a certain size and color. We also learned how to apply the concept of potential field to make a robot that avoided obstacles.

Going forward, I will remember the importance of leaving enough time to test and how to stick to a list of priorities.

VII. How to do stuff

Its appearance should be as close to this document as possible to achieve consistency in the proceedings.

References should be cited as numbers, and should be ordered by their appearance (example: "... as shown in , ..."). Only references that are actually cited can be listed in the references section. The references' format should be evident from the examples in this text.

References should be of academic character and should be published and accessible. Your advisor can answer your questions regarding literature research. You must cite all used sources. Examples of good references include text books and scientific journals or conference proceedings. If possible, citing internet pages should be avoided. In particular, Wikipedia is *not* an appropriate reference in academic reports. Avoiding references in languages other than English is recommended.

Figures and tables should be labeled and numbered, such as in Table IV and Fig. 4.

TABLE IV: Simulation Parameters

Information message length	$k = 16000$ bit
Radio segment size	$b = 160$ bit
Rate of component codes	$R_{cc} = 1/3$
Polynomial of component encoders	$[1, 33/37, 25/37]_8$

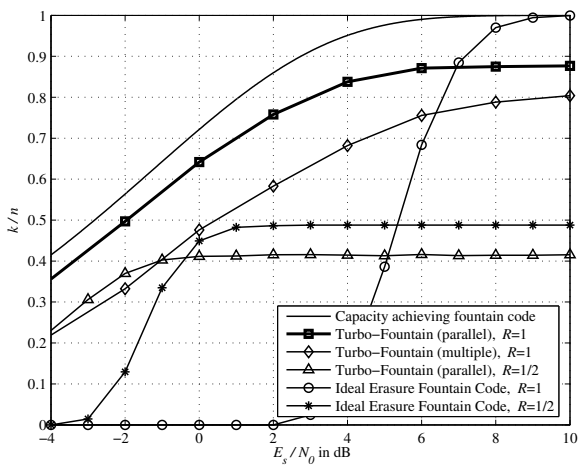

Fig. 4: Simulation results on the AWGN channel. Average throughput k/n vs E_s/N_0.

VIII. Filling this page

Aallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua

Celtae, nostra Galli appellantur. Gallos ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit. Horum omnium fortissimi sunt Belgae, propterea quod a cultu atque humanitate provinciae longissime absunt, minimeque ad eos mercatores saepe commeant atque ea quae ad effeminandos animos pertinent important, proximique sunt Germanis, qui trans Rhenum incolunt, quibuscum continenter bellum gerunt. Qua de causa Helvetii quoque reliquos Gallos virtute praecedunt, quod fere cotidianis proeliis cum Germanis contendunt, cum aut suis finibus eos prohibent aut ipsi in eorum finibus bellum gerunt. Eorum una, pars, quam Gallos obtinere dictum est, initium capit a flumine Rhodano, continetur Garumna flumine, Oceano, finibus Belgarum, attingit etiam ab Sequanis et Helvetiis flumen Rhenum, vergit ad septentriones. Belgae ab extremis Galliae finibus oriuntur, pertinent ad inferiorem partem fluminis Rheni, spectant in septentrionem et orientem solem.

IX. Conclusion

This section summarizes the paper.

References

[1] M. T. Boulet. RACECAR: Rapid autonomous complex-environment competing ackermann-steering robot. http://peris.mit.edu/racecar/lectures/2015/Lecture2.pdf, July 2016.

[2] Michael S. Triantafyllou and Franz S. Hover. Maneuvering and control of marine vehicles. MIT Department of Ocean Engineering, November 2003.

[3] Friday challenge: Make the correct turn. MIT Lincoln Laboratory Beaver Works, July 2016.

[4]

[5] A. Anders, M. Boulet, and S. Karaman. mit-racecar. https://github.com/mit-racecar, 2016.

[6] http://www.ieee.org/web/publications/authors/transjnl/index.html. IEEE Transactions LaTeXand Microsoft Word Style Files.