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 compete in the Mini Grand Prix Challenge, an autonomous car race around a track. This program, which is based off the 6.141 Robotics Science and Systems class at MIT and the Robotics IAP, meets on weekdays for eight hours for four weeks. Each day typically consists of three lab periods, a technical lecture, a seminar with 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 experiment with them on a 1/10-scale racecar. Teams could explore the abilities and limitations in building robust yet fast robotics systems. Some of the tasks given include following a wall, detecting a colored blob, following a colored blob, reactive obstacle detection, using a color to decide whether to take a turn, and optionally implementing localization and mapping. Much of the class dealt with how to deal with problems inherent to autonomous navigation, like sensor imperfections, 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 includes the NVIDIA quad-core CPU, the 192-core GPU, an Ackermann-steering system, and a large variety of sensors including a Lidar, 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 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 starting at various points away from the wall.

B. Approach

The teams were required to use a Lidar to sense the wall but they were free to implement a state controller of their choice. [Go into details about state controllers]

1) Bang Bang Controller: The simplest of the methods was using a Bang Bang controller, which would sense if the robot was too close or too far and if it was too close, it would turn away and if too far, turn towards.

2) PID Controller: A slightly more sophisticated version was to implement a PID(Proportional Integral Derivative) controller which does calculations on the error value to create a smooth oscillation to correct for the robots distance. In addition, for 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.

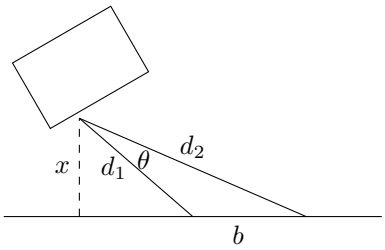

Fig. 1: Diagram of robot

A two point controller could account for this with basic trigonometry. Given two arbitrary angles to take the laser measurements d_1 and d_2, if the angle θ is known the following formula can calculate the true distance between the wall.

$$x = \frac{d_1 d_2 sin(\theta)}{\sqrt{d_1^2 + d_2^2 - 2d_1 d_2 cos(\theta)}} \tag{1}$$

Our team however 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.

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. We also decided to approach the challenge by starting simple and incrementally improving it.

C. Process

We started with the bang bang controller and as that worked, we slowly added a PID controller with 2-point control. For the bang bang controller, it was just a very simple algorithm that detected the amount of error in the cars distance from the wall.

If the error was too big, it would turn towards the wall, and if the error was too small, it would turn away from the wall. The bang bang controller worked well to turn at a moderately steep curve.

As seen in the video, it worked well despite being a little jerky. 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 back 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 inefficient oscillations and jerkiness.

We next tried implementing a P controller in order to control the oscillations 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. It took a while to figure out the Kp value. With higher Kp values, we found that we could adjust the rate of responsiveness. We found that Kp values near 1.0 achieved the best balance of reactiveness but not overreaction.

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. We first found that changing the Kd value would lower the optimal Kp value. After a lot of thorough tuning of the Kp and Kd constants, we found that incorporating derivative caused jerkiness no matter what value we had it at. Although it was jerky we also found that it was necessary for adjusting when really far. When we tried the P controller and high distances, it would sometimes turn back so quickly that it would smash into the wall. With the PD controller, we found that even though the execution was shaky, it allowed the robot to not crash into the wall. Then we thought that there was no point of tuning the PD controller if we had to retune for the PID controller so we tried testing the PID controller. In the end, a large majority of the challenge had to do with finding the right constants. The setup that ended up working well was with a Kp of 1, a Kd of .05, and a Ki of 0. With more time, we could have continued perfecting the numbers, but they were sufficiently good for this challenge [research values for P, I, D]

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. We learned that although there is a lot a theoretical research on optimal values for the PID controller, what happens in real life is vastly different. A lot of testing is required. According to Kyle from JPL, although the guess and check method was tedious, it was much easier than the alternative. By week one, we got a good idea of this problem in robotics, that what should happen or what happens in simulation is not a substitute for real life testing. Our race 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 got 8th out of 9 teams, the times were all within .2 seconds of each other so it could have been external factors from our algorithm such as battery level or timer inaccuracy. That being said,with more time the constants could have been more tuned to a more optimum 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.

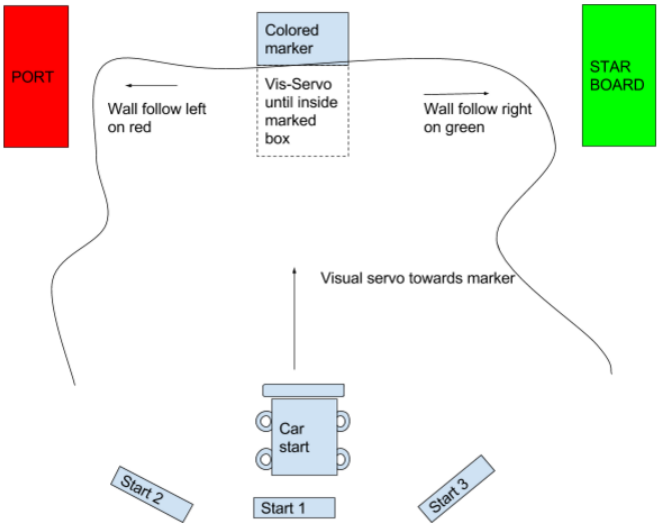

Fig. 2: Challenge diagram [2]

B. Approach

We started with getting a good blob detections algorithm, then worked on a simple but effective visual servoing method that could then transition into the wall following modified from last week. We split up the task into a blob detection node, 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 the image detection, then split people up to work on various aspects of the visual servoing and wall following. 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. We split up the team, giving different people different tasks. Some members of the group worked on a visual servoing method using a bang bang method based on height and x position of the blob. If the blob was too far it would go forwards, if it was close enough it would stop. If the x position of the blob was on the right side of the screen, the robot turned right. 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.

While some people were working on that, others worked on taking our wall-follow code from last week and modifying it and optimizing it for our situation. We didnt focus on tuning this part too much because we knew it functioned from last week. Another sub-group worked on setting up the structure of the task and how the nodes would communicate. 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. 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. Importance of testing

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.

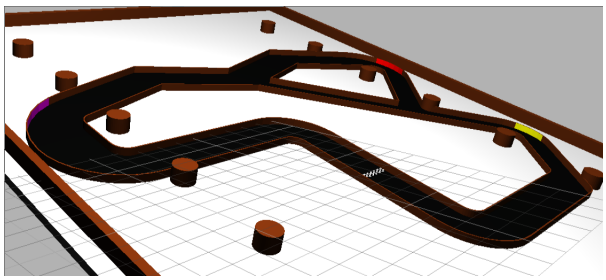

Fig. 3: Model of racetrack [3]

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	$[1, 33/37, 25/37]_8$

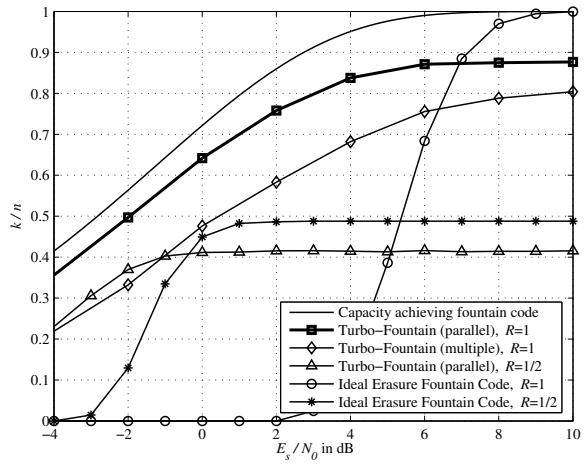

Fig. 4: Simulation results on the AWGN channel. Average throughput k/n vs E_s/N_0.

VIII. Filling this page

Gallia 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] Friday challenge: Make the correct turn. MIT Lincoln Laboratory Beaver Works, July 2016.
[3] A. Anders, M. Boulet, and S. Karaman. mit-racecar. https://github.com/mit-racecar, 2016.
[4] http://www.ieee.org/web/publications/authors/transjnl/index.html. IEEE Transactions LaTeXand Microsoft Word Style Files.