Oh, Braden, Author. *Data Architect, Team 4*

Development and Implementation of Autonomous RACECAR Systems

Abstract – At the four-week Beaverworks Summer Institute, high school students were taught the basics of autonomous vehicle design and implementation. Students learned the basics of control algorithms (specifically bang-bang and PID control), methods of computer vision in identifying colored markers, and basic localization and mapping algorithms for obstacle avoidance. The final week was spent preparing for a final challenge that forced students to implement a complete autonomous system on a dynamic race course. Our team implemented a potential field control system with free-space vector overlay and a blob detection color system to place 5th overall in the final race.

I. Introduction

BEAVERWORKS is a research cooperative founded by Lincoln Laboratory and the MIT School of Engineering with the purpose of facilitating research and innovation[1]. In an effort to expand beyond the university classroom, the institute offered its first four-week intensive study for high school students in the field of autonomous vehicle programming and design in the summer of 2016. The program's curriculum was adapted from MIT RACECAR, an optional, four-week course offered to university students in-between semesters at MIT. The Beaverworks program taught students the fundamentals of autonomous steering algorithms, computer vision, and computer planning while allowing them to implement their systems on miniaturized autonomous vehicles. Each of these core fields was taught during one of the first three weeks; students spent the fourth week preparing an autonomous system for competition in a final race.

II. Week 1

The first week was spent familiarizing students with ROS (Robot Operating System), a popular control system for robotics hardware, and teaching the bang-bang and PID (Proportional, Integral, Derivative) steering control algorithms. The week culminated in a two-lane drag race where students raced their cars along a straight wall with only the steering algorithm in control.

A. The RACECAR Platform

The MIT RACECAR (Rapid Autonomous Complex-Environment Competing Ackermann-steering Robot) is an autonomous vehicle platform designed for student learning and development[2]. The platform is based upon a 1/10 scale Traxxis RC rally car and sports an Nvidia Jetson TX-1 embedded supercomputer with a 4-core CPU and 256-core GPU (graphics card). For sensors, the car sports a Hokuyo laser range finder (LIDAR), Stereolabs ZED camera, and Sparkfun IMU (Inertial Measurement Unit). The car is also able to support a Structure.io depth camera, though the depth camera was removed for this course (rendered redundant, as the stereo camera is capable of interpreting depth). As a control mechanism, the entire vehicle was equipped with an open source ESC (Electronic Speed Controller) called VESC, which doubles as an odometer. Hardware specifications and software components are available publicly through github[3]. As an operating system, the Jetson computer runs Ubuntu 14.

Fig. 1-1. An MIT RACECAR as used in summer 2016. This vehicle does not use the structure.io depth camera and has a router mounted directly to the top of the chassis.

August 2016. This work was facilitated by Beaverworks, an associate institution of Lincoln Laboratories and the Massachusetts Institute of Technology's School of Engineering.

Braden Oh, Author, is a rising senior at La Cañada High School, CA USA (e-mail: braden.oh@ icloud.com). As of August 2016, he is with MIT KitCube, a team based out of the MIT AeroAstro Department, Cambridge, MA USA and is in the process of applying to college.

B. ROS

The RACECAR system is complex, capable of simultaneously collecting data from numerous sensors, performing computations on that data, and issuing speed and steering commands to the VESC. In order to facilitate so many parallel processes, the system is controlled by the open-source Robot Operating System (ROS). ROS Indigo, a version of ROS released in July of 2014, is used on the RACECAR as it is optimized for Ubuntu 14[4]. In order to facilitate data-handling from many simultaneous processes, ROS features a system of "nodes" that "publish" and "subscribe" to various data "topics." The terminology is as follows:

- *Nodes* are pieces of software, usually written in Python or C++, that perform a particular task or computation
- *Publishing* data is a method of packaging and sending information to a data thread. Nodes publish data to topics.
- *Topic* is synonymous with data thread
- *Subscribing* is a method of reading in data packets from a topic. Nodes subscribe to topics in order to read the data published there

Students spent their first days at Beaverworks becoming familiar with ROS by writing simple safety nodes that could stop a car if it drove too close to an object; LIDAR data was interpreted by a node that published distances to the topic "/scan." The safety node (subscribed to the /scan topic) analyzed the data, then published a drive command to the topic "/teleop." The drive system (subscribed to /teleop) froze the vehicle when the safety node published a speed-zero command. Once these safety scripts were written, students were ready to start building autonomous steering controllers.

C. Bang Bang Control

The simplest method of controlling a robot is the bang-bang controller[5], a binary steering system which operates by abruptly switching between two states[6]. As a steering algorithm, the bang bang controller attempts to follow a path by switching between steering full left and full right when not directly on course. During week one, these controllers were used to follow a wall. The RACECAR was given a nominal distance to stay from the wall (usually ~0.5 meters) and used LIDAR data to check that horizontal distance. When the LIDAR indicated that the vehicle was too close (<0.5m), the car turned full right to compensate. When the LIDAR indicated that the vehicle was too far (>0.5m), the car turned full left.

The major advantage of bang-bang is its simplicity[7]; there are few lines of code and the algorithm is simple, so errors are easy to diagnose and debug. This makes the bang-bang controller a highly effective means of achieving a certain degree of control. However, there are two major disadvantages that render bang-bang dangerous to use. The first is oscillation; being that the algorithm can never drive forwards (only left or right), the vehicle follows a sine-curve-like path whose center line is located at the desired distance from the wall. As a result, the car is rarely ever at the desired distance. The second major disadvantage is overcompensation; when the car is placed a high distance away from the wall, the aggressive steering can cause the car to turn too far to be able to react to the wall; by the time the car finally passes the 0.5m threshold, it may be facing the wall head-on or even facing backwards. In an even more extreme case (where the wall is >1.5m away), the car will not ever pass the 0.5m threshold and will drive in circles.

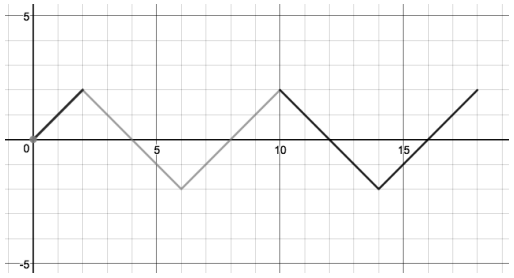

Fig. 1-2. The path followed by a bang-bang controller attempting to center itself on the x axis

The algorithm our team ended up implementing was unable to solve the overcompensation issue, though *was* able to help mitigate the oscillation; if the vehicle was within 10cm (0.1m) of the nominal distance, it would issue a drive-forwards command, rather than a left or right command. In this way, the car spent a fair amount of time driving forwards, avoiding a portion of the oscillation inherent in a binary system.

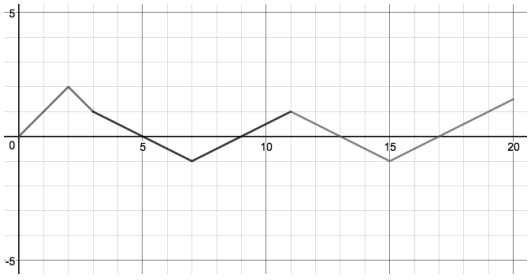

Fig. 1-3. The path followed by a bang-bang controller modified to drive straight part of the time. The oscillation present is due to a combination of inherent drift in the car and steering overshoot

D. P Control

The second simplest method of controlling a vehicle is the proportional controller[5], or P controller[8], so named because it outputs a steering command proportional to its distance from the desired target. This calculation can be performed by the equation

$$\theta = Kp \cdot e$$

where the outputted steering angle, θ, is determined by the amount of error, e, multiplied by an experimentally-derived constant, Kp. In its simplest form, Kp = 1, and so the steering angle is equal to the amount of error present (thus completely proportional; as error increases, so does the steering angle and vice versa). This approach does not often work, however, as the amount of error can be extremely large and so creates a disproportionate steering angle to what is necessary (i.e. The car is facing 60° off course, so sets the steering angle to 60°, and ends up facing perpendicularly to the line it means to follow).

By varying the value of Kp (usually | {0 < Kp < 1}), the steering angle can be "tuned" to respond to the amount of error in a more useful way (i.e. The car is facing 60° off course, so sets it steering angle to 6°, allowing it to drift back towards the line). In this workshop, the value was experimentally derived, and our team found that a Kp value of 0.1 produced an optimum controller. Other teams' Kp values varied, some as low as 0.01 and others as high as 0.8, indicating that hardware plays a large factor in the outcome of a controller's effectiveness.

The P controller alone, however, often leads to an unstable system; A vehicle placed away from the line it hopes to follow may approach the line, and then overshoot. As it overshoots, the controller realizes that the vehicle is (significantly) off course and will turn back towards the line with a proportionally significant response. This in turn creates an even larger overshoot, and the system collapses into increasing oscillations.

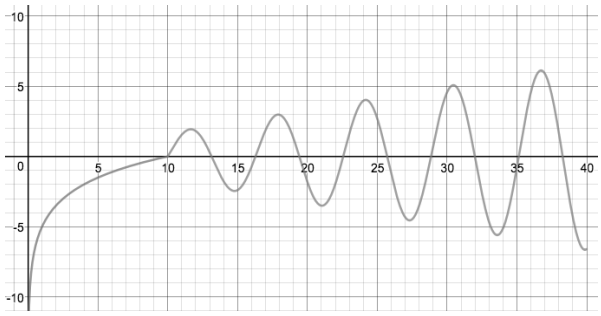

Fig. 1-4. The increasingly oscillating path followed by an unstable P controller

E. PD Control

In order to help prevent these oscillations from perpetually increasing, a derivative (D) term can be added to the proportional (P) term to create a PD controller[8] with the equation

$$\theta = (Kp \cdot e) + (Kd \cdot \dot{e})$$

where the outputted steering angle, θ, is determined by the P term summed to a D term where Kd is an experimentally-derived constant (that differs from Kp) multiplied into the derivative of the error at that point, ė. Kd is set | {Kd < 0} so that a resultantly negative D term dampens the positive steering angle set by the P term; if the amplitude of oscillations is very large, the secant (derivative term) between the two time intervals is proportionally large. Subtracting this secant term from the P term reduces the resultant steering angle. Since a derivative is proportional to the amplitude of the oscillations it is derived from, the D term effectively dampens an unstable P system by reducing the resultant steering angle proportionally to the size of the instabilities.

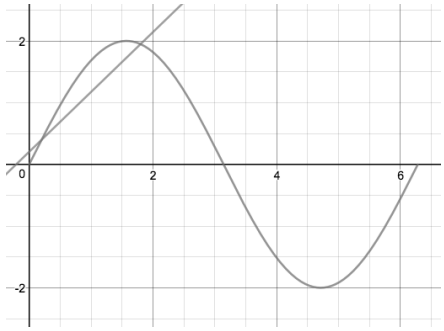

Fig. 1-5. If the amplitude of oscillations is very large, the secant between the two time intervals is also very large. Subtracting this secant (the horizontal line) from an unstable P controller (the sine curve) produces a lower net steering angle, thus dampening the effect of oscillations.

For purposes of programming the vehicle, a secant line sufficed for the ė term and was calculated by the equation

$$\dot{e} = previous\ error - current\ error$$

Like with the Kp constant, the Kd constant required tuning. Our team found that a tiny Kd value worked well, and used Kd = 0.05 during the week's challenge.

F. Determining Distance

While having effective controllers is important, each of these control algorithms rely on obtaining an accurate perpendicular distance to the wall. This task is not easy because the car is not always oriented parallel to the wall. In the best case scenario, the car is located parallel to the wall, such that the point at 90° from the front of the car is the accurate position of the wall.

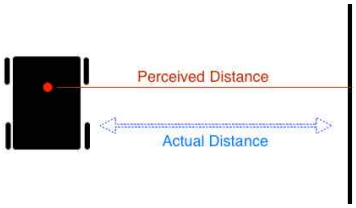

Fig. 1-6. If the car is oriented parallel to the wall, the laserscan at 90° (perceived distance) is equal to the actual distance, so provides an accurate measurement

In the case that the car is oriented away from the wall, the distance at 90° will read greater than the correct distance, and the car will shift its wheels towards the wall.

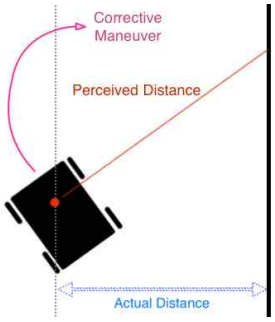

Fig. 1-7. If the car is oriented away from the wall, it perceives its distance from the wall to be too large and will swerve towards the wall. This maneuver can sometimes be helpful

This is not inherently a bad thing; in fact, it can often help keep the wheels oriented forwards. In the case that the car is oriented towards the wall, however, this error can cause the vehicle to crash. If the car perceives itself to be to far from the wall it will attempt to drive towards the wall; if the car is already oriented towards the wall when this maneuver occurs, the car will crash.

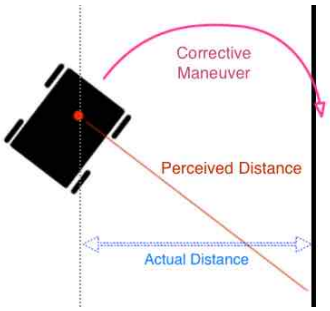

Fig. 1-8. If the car is oriented towards from the wall, it may still perceive its distance to be too large and will swerve into the wall.

Solving this problem can occur in two ways, both of which our team experimented with during the institute. The simplest method is to parse through all the laserscan data on the side of the wall and look for the shortest distance; the shortest distance from the LIDAR scanner to the wall is always the perpendicular. This method, while quick and easy, may be prone to errors in the LIDAR data (the laser scanner is not inherently accurate).

A more robust solution to the distance problem is to read not only the point 90° from the front of the car, but also a point 30° off of that point; in that way, the car has generated an SAS triangle, a triangle with two adjacent sides and the included angle (side-angle-side) all known values.

From here, the law of cosines can be used to calculate the third side of the triangle and thereby the area. The equation area = base • height can then be used to back-calculate the perpendicular to the car, which will always be the height of the resultant triangle.

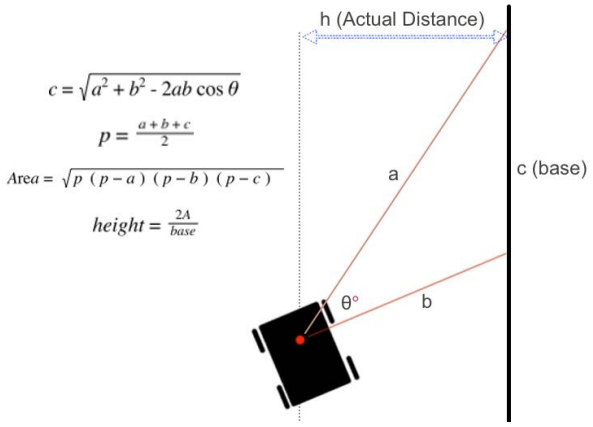

Fig. 1-9. The algebraic and trigonometric processes by which the true distance from the wall to the car can be derived by two points

Our team implemented the robust (trig) method during week 1, but experimented later in the course with simply taking the closest point in the range; while this second method is theoretically less robust, it has not proven to be unreliable for purposes of the RACECAR.

G. Week 1 Challenge

The challenge ending week 1 was a wall-following drag race; a car was placed on an ~1.5 meter wide, straight-walled track bounded by ~1/3 meter high cardboard fluting. The car was required first the left and then the right wall using LIDAR data and some sort of steering controller. Our team opted to use a PD controller with Kp and Kd values of 0.1 and 0.05, respectively. The error term was defined as the desired distance from the wall minus the actual distance from the wall (as determined by the LIDAR). The desired distance our team set was 0.5 meters from the wall.

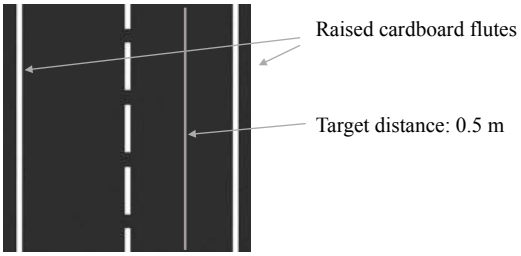

Fig. 1-10. Raised cardboard flutes were used to bound both sides of the straight track. A tape line down the center marked left and right lanes. The desired distance our team chose from either wall was 0.5 m. *Image credit: www.gograph.com*

Our team successfully followed both the right and left walls without collisions, but when the car was placed very far from the right wall, it was unable to correct its course in time to avoid colliding with the wall; the P controller forced the car very quickly towards the wall (being that the car's initial distance was very far from the wall) and ended up overshooting the 0.5m target. line The P controller then realized that it had overshot the line and attempted to create a large steering angle away from the wall. We presume that our Kd value was too large, as it too effectively dampened the steering angle; rather than turning back quickly enough, the large derivative prevented the car from creating a large net steering angle, and so it ended up colliding with the cardboard fluting.

III. WEEK 2

The second week was spent teaching students the basics of computer vision; they read in images with the onboard ZED camera and utilized image processing functions built into the Python OpenCV library. The week culminated in a challenge where the RACECAR encountered a colored marker, steered towards it relying only on vision (no LIDAR), and then, based on the marker color, turned a certain way at a T-junction.

A. OpenCV

Open Source Computer Vision (OpenCV) is a library of image processing functions with interfaces for Java, Python, C, and C++. It was built and optimized in C-based languages and is designed for computational efficiency[9].

B. RGB and HSV

One of the early challenges was to detect a colored marker; this is not particularly easy when using the standard RGB (Red-Green-Blue) color scheme, as varying brightnesses and intensities of light are difficult to isolate by nature of the three distinct variables. As a result, the image analysis performed by our team was executed in the HSV (Hue-Shade-Value) color scheme. In HSV, color in the traditional sense is defined on a 360° color wheel independent of brightness or color intensity, which are represented as value and shade, respectively[10]. In this way, colors are easier to isolate in computations; all of yellow can be represented by a single range of hue values (45-75) and bright shades can be isolated efficiently by simply choosing large shade and brightness values. Color identification was by far the most difficult part of the week for our team, as the HSV values for the colored marker targets had to be manually changed and tested and any variation in lighting or shadow rev

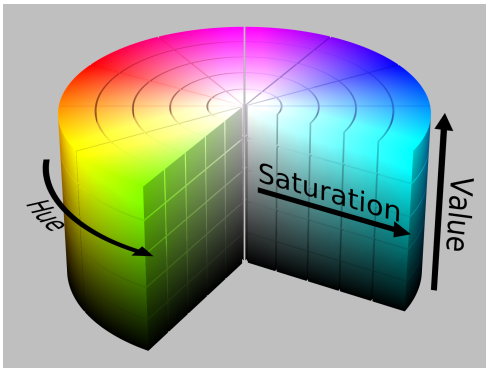

Fig. 2-1. A 3D representation of the HSV color scheme; colors are independently isolated around the hue color wheel, intensity of color by saturation and brightness by depth.
Image credit: www.wikimedia.com

C.Blob Detection

OpenCV features a variety of "blob detection"

methods; the <SimpleBlobDetector> class has built-in functions for identifying and filtering objects in an image based upon color, area, circularity, ratio of inertia, and convexity[11].

The first step in blob detection is to read in an image; on the RACECAR, images were provided by the left camera of the stereo ZED camera. In order to detect a colored marker, a bitmap was extracted from the imported media; the image was first converted from RGB to HSV and then had the pixels of a certain HSV range isolated in a bitmap. The SimpleBlobDetector functions were utilized on this bitmap to detect objects.

Being that the colored marker was large (8.5" x 11"), we were able to detect the marker by filtering blobs by size; the largest blob on screen was assumed to be the colored marker (being that there were no objects in the room with both a comparable color range and area to the markers).

D. Week 2 Challenge

The challenge ending week 2 was to navigate a T-junction, with a colored marker was deployed at the center of the T. This marker signified the direction the car should turn in (red for left, green for right). The car would then wall-follow along the corresponding path, marked by cardboard flutes.

In order to make the challenge more complex, the path towards the marker was not marked and the cars were started without directly facing the marker (the marker was always in the camera's field of motion, but seldom head-on). As a result, a method for controlling the car towards the colored marker was necessary.

Furthermore, no LIDAR data was allowed to be received until the car entered a region near the colored marker, marked on the floor by yellow tape. Thus, approaching, interpreting, detecting the distance of the wall, and detecting the color of the marker had to have been performed by the camera alone.

Our team solved this problem by extrapolating additional data from the bitmaps extracted from the ZED images. Once the marker was isolated, additional OpenCV SimpleBlobDetector methods were used to identify the center of mass and width of the marker. From this information, our team was able to derive the center point of the marker, a target point that we could use in a steering controller. Recycling code from the PD controller, our team used this center point as the target, (rather than the distance from a wall), granting us the ability to home in on the colored marker.

Being that the vehicle was only reading in data from the left ZED camera (thus having no stereo vision for depth), our team resorted to a simple method for determining when the car was in the box; since the box was located very close to the marker (and the car was navigating towards the marker), we reasoned that once the marker's (unique) color filled a certain percentage of the camera frame, it would be far enough inside the box to be able to enable its LIDAR sensor. To reduce the number of pixels (and colored objects) in the frame, our code cropped all camera images to 60% height and programmed the car to activate its LIDAR scanner when the colored marker filled 11% of the resultant frame.

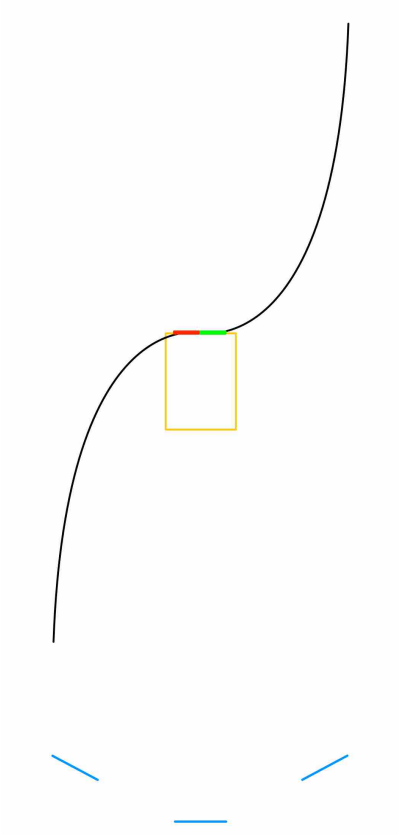

Fig. 2-2. A bird's eye view of the week 2 challenge course. The car was positioned at one of the three starting locations (blue) with a field of vision including the colored marker. The colored marker (red or green) denoted which way the car should turn (left or right, respectively). The car had to navigate into the marked box (yellow) before activating its LIDAR, after which it would wall follow along a curved path marked by the cardboard flutes (black).

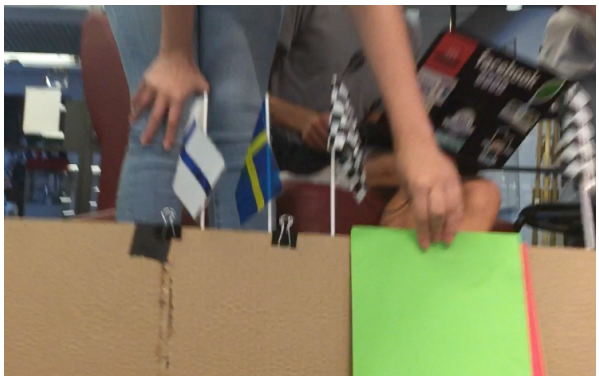

Fig. 2-3. This image was taken from the vehicle as it approached the colored marker; once the marker filled more than 11% of the frame's volume, the car activated its LIDAR scanner

Our team was extremely successful during the final challenge; our approach and LIDAR activation systems worked reliably and our car was successfully able to start wall-following in both directions. The most difficult part, by far, was tuning the PD controller for the right-hand track; ~1m from the cardboard fluting on the right side was a narrow steel post supporting a handrail. This obstacle was not marked on the map and did not affect vehicles that hugged the wall during the wall-follow; however, due to the steep curvature of the wall, following the wall at a sufficiently close distance was incredibly difficult.

Our team eventually overcame these issues and was one of only two teams to successfully complete the challenge. Video of our team's successful run can be found online:

https://youtu.be/DUp9yURMo2c

You will easily notice the instability of the PD controller in the shakiness of the video; this instability was unfortunate, but was the only way to avoid crashing into the steel pole. We came in second place overall, as the first place team had completed a run slightly faster than ours had.

IV. Week 3

The third week was spent teaching students obstacle avoidance algorithms and autonomous localization and mapping. Students used ROS mapping algorithms to develop maps of an enclosed space and implemented potential field systems for obstacle avoidance. The week culminated in a blob detection challenge within an enclosed space, where cars were given 2 minutes to explore an enclosed space, counting as many colored markers (of varying colors) as they could.

A. Localization

In order to move through an environment effectively, an autonomous system should know where it is in the environment. Localization is a complex set of tasks that attempts to do just that.

Dead reckoning is a simple method of localization that relies on odometry data to estimate the vehicle's current position and orientation[12]; the vehicle defines its starting position as the origin in a cartesian coordinate system and then tracks its position within that system by counting wheel revolutions and their corresponding directions. Odometry data, however, is often unreliable, especially so with the RACECAR hardware; the RACECAR has no odometry sensor, all 'odometry' data comes from estimations made by the computer based upon the voltages sent to the motors and actuators.

A more reliable form of localization relies on sensor data and a system of landmarks[13]; a sensor (i.e. LIDAR scanner or camera) observes its surroundings and identifies two distinct points (landmarks) ahead of it. The vehicle then drives forward and checks the new position of the landmarks, using the new positions relative to the vehicle to calculate its position in the environment. The vehicle then chooses two new points ahead of it and the process begins again.

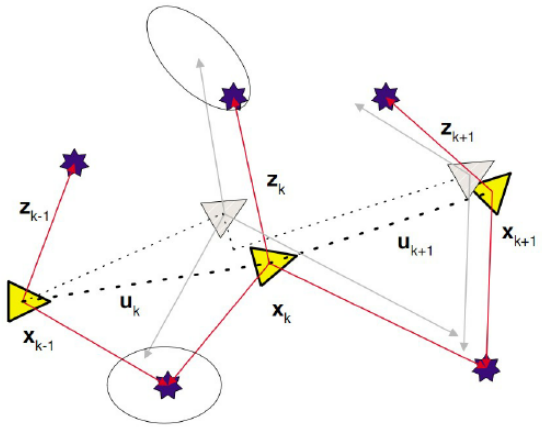

Fig. 3-1. An autonomous vehicle (X_{k-1}) identifies two points ahead of it (Z_{k-1} and X_{k-1}), moves forward to position X_k, checks its position relative to those points (now X_k and Z_k) to triangulate its current position, then chooses new points ahead of it and begins the process again. *Image credit:* www.researchgate.net

Assuming that the environment is static, this localization algorithm is far more reliable than odometry as it allows the system to correct for error in real time. In Fig. 3-1, the grey 'ghost' images under X_k and X_{k+1} as well as the red circles around the star

objects represent inherent error; sensors may be imprecise at identifying the locations of objects (red circles) and steering/odometry may be imprecise at moving the vehicle to a desired point (ghost images) relative to those objects. The constant re-checking and derivation of position by referencing static objects allows the vehicle to verify its position at every instance it checks its surroundings.

B. Mapping

Suppose an autonomous vehicle is exploring an environment, reading in LIDAR data as it goes. It would be invaluable to the autonomous system to log its environment as it goes such that it knows where it is in a global reference frame. The ROS gmapping package provides a set of laser-based mapping algorithms that can, in real time, build build and return a 2D occupancy grid map similar to a building floor plan[14]. This map can be analyzed, in turn, by ROS nodes and utilized in robot planning and memory algorithms.

As an autonomous vehicle drives around, it reads data into its LIDAR scanner. This data is returned in the form of a 1D point cloud listing distances to objects in a 270° arc. As the data is read in, the gmapping package marks the points detected in a 2D grid map, effectively "drawing" detected points in a plane for later reference. The resultant grid map can be analyzed by an external ROS node.

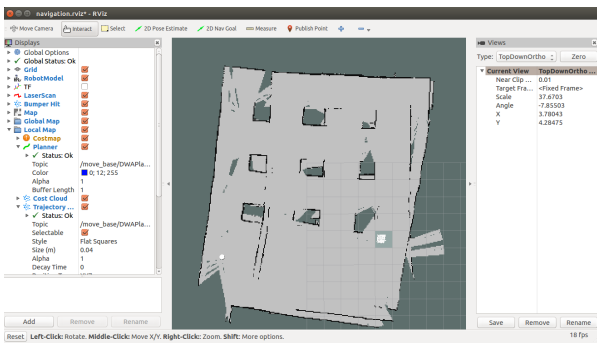

Fig. 3-2. A screenshot of the ROS Gazebo visualizer in the process of building and reading a map created by the ROS gmapping package and a laser scanner.
Image credit: www.hackaday.io

C. SLAM

Suppose an autonomous vehicle is moving through an environment, identifying colored objects that it passes by. Without being able to identify where in the world it is, objects could be re-counted as the robot passes by them a second time. An effective way to overcome this issue is to have the vehicle map the environment it is passing through as it goes. However, the robot would be unable to tell if it is in a given spot on the map it has drawn without being able to tell where it is in reference to the map. As a result, mapping and localization must happen simultaneously.

This is known as the Simultaneous Localization And Mapping (SLAM) problem; as the robot passes through a world, it draws a map. Simultaneously, it uses the the same points it uses to draw the map to localize its position with respect to its surroundings. By comparing the points around it with points in the map, a vehicle is able to approximate its location in a global reference frame[15].

D. Object Avoidance

Once a robot is able to solve the SLAM problem, the idea of object avoidance becomes much easier; by referencing a map of the environment, a vehicle is able to find potential obstacles and then is able to identify their positions relative to itself via localization. It can then plan paths that do not coincide with these obstacles. There are a number of methods for achieving SLAM-based object avoidance, but the issue our team faced was that they all involved the use of a map. Being that we would not receive a map of the environment in advance, we set out to design an alternate set of algorithms that did not require the use of a map.

E. White-Space Avoidance

A LIDAR scanner returns a 1D array of numbers, each number representing a distance and each position in the array representing a point along a 270° arc. An easy method of filtering through the data is to parse through the array, searching for values that are within some distance from the car (i.e. search for all points that are <5m from the vehicle). The results of this search can be processed into a bitmap of ones and zeros, ones representing the position of objects within the threshold distance and zeros representing usable white space.

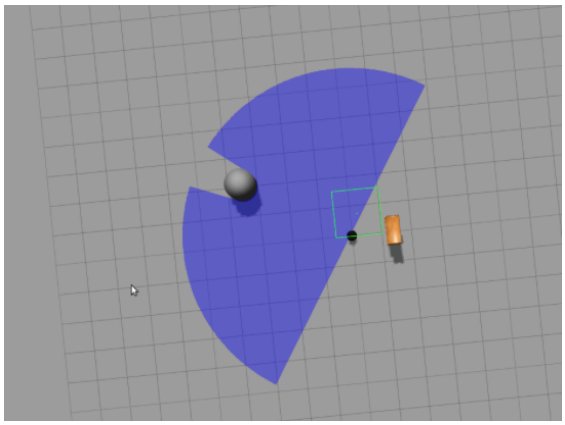

Fig. 3-3. A screenshot of the ROS Gazebo visualizer simulating laserscan data. *Image credit: www.ros.org*

Assume the vehicle in Fig. 12 is reading in data every 10° from the 180° arc represented by the purple field. Also assume that the laserscan has a radius of 5 meters. If the object in the frame is only 4 meters away from the object, and its searching threshold is 5 meters, it might return data in an array such as this:

[5, 5, 5, 5, 5, 5, 5, 5, 4, 3, 4, 5, 5, 5, 5, 5, 5, 5]

Where each data point represents the range from the laser scanner to the nearest object in its 10° window. Being that the obstacle is cylindrical, the scanner, in this instance, detects the edges of the obstacle at positions 9 and 11, and the center of the object at position 10. After searching for all values <5m away, the new bitmap array would look like this:

[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

Where each 0 represents at least 5m of free space and each 1 represents an obstacle. This bitmap can be easily visualized by projecting empty space into every point where there is a 0 and a bar into every point where there is a 1:

[——]

This representation of the environment is primitive but effective; by simply identifying and moving towards the largest section of white space, the car knows where no obstacles exist 5m ahead of it. As it moves forward, into this white space, it repeats the scanning process to identify new obstacles that exist ahead of the new position.

Our team successfully implemented a prototype of this white-space system that worked quite effectively. Being that the only PD controller required was to steer the car towards the center of the largest section of white space, the number of variables that required tuning was minimal, and the vehicle was able to move through a space with obstacles without collisions. Our team did not implement thorough testing, however, especially for corner cases, as we soon moved on to more sophisticated methods of object avoidance.

F. Farthest-Vector Avoidance

An even simpler approach to white-space avoidance was proposed to our team by Oktay Arslan[16], a specialist in robotics from JPL who gave a set of guest lectures to the institute. Utilizing the fact that the LIDAR effectively has no maximum range, he proposed an algorithm that simply searches for the farthest distance away from the vehicle and drives toward that point. Our team toyed with the idea for a little while, but were concerned by how easily it might be confused; in the final race, without obstacles, the algorithm might work reasonably effectively. However, the weekly challenge asked for object avoidance and featured closely-placed objects, some of which had gaps between them; being that the farthest vector could potentially lie between the two obstacles, but without enough clearance for the vehicle, our team did not want to risk the chance of a collision so did not implement this approach during week 3.

G.Potential Field Avoidance

Gravity is described as projecting a force field, a field of *gravitational* potential. Similarly, positive and negative charges project force fields, fields of *electric* potential. An alternative technique to white-space techniques lies in creating a vector space with behaviors like that of a potential field.

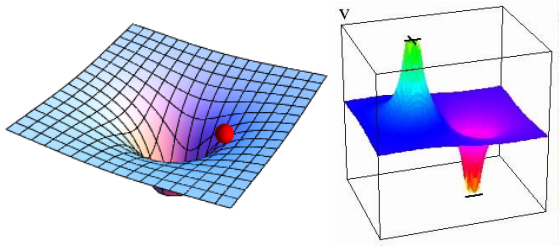

Fig. 3-4. A planet (left) and an electric charge (right) both create large regions of low or high potential in their corresponding fields. The moon (red ball) wants to roll towards its planet and charges want to move from the high positive region to the low negative region. This property of a potential field can be harnessed for object avoidance.

Image credit: Greg Egan (left) and UC Davis (right)

In considering potential fields' application to object avoidance, it is easiest to consider electric potential. The electrical force between two charges is calculated by Coloumb's Law[17], given as

$$F = \frac{kQq}{d^2}$$

where the resultant electric force between two particles (F) is equal to the electric constant (k) times the charges of the two particles (Q and q) divided by the distance (d) between them squared. This equation is known as an inverse-square law, being that the force is inversely proportional to the square of the distance between the two particles. This means that as d increases linearly, F increases exponentially.

Consider an autonomous car to be positively charged and for each object the LIDAR detects to also be positively charged; the car will be repelled from every object it detects. The strength by which it is repelled by a given object can be determined by Coloumb's Law. This results in that as the vehicle approaches an obstacle, it is exponentially repelled. In order to get the car to move forward, consider a strong, permanent positive charge to always exist behind the car. In this way, the car passes through a world, repelled by everything, seeking the path of lowest potential.

This algorithm is surprisingly easy to implement; vectors are easy to compute and manipulate in Python, especially with NumPy libraries[18]. A 1D array of data points is read in by the LIDAR scanner and each point is passed through a modified form of Coloumb's Law, given as

$$F = \frac{K}{d^2}$$

where K is an experimentally-derived, easily tuned constant and d is each range value in the scanner array. Passing the scanner array through this equation produces an array of magnitudes – the "strength" by which each point in space "pushes" on the car.

The angular direction of each magnitude is known by its position in the array. As such, simple trigonometry gives each of these direction-magnitude vectors a corresponding Cartesian x and y magnitude. Summing together all x magnitudes (some of which point left and some of which point right) results in a net x value, X. Summing together all y magnitudes results in a net Y value.

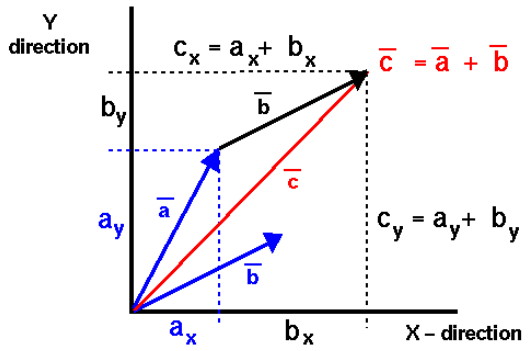

Fig. 3-5. Summation of vectors; the x and y components of vectors a and b, when combined, create a net X and net Y that represent the vector a+b. *Image credit: NASA Glenn*

The car's speed is thereby represented by Y and its steering direction by X. The summation and trigonometric processes for the entire calculation of X and Y can each be expressed in a single line:

$$X = \sum_{d[0]}^{d[n]} \frac{K}{d^2} \cdot \cos\theta$$

$$Y = \sum_{d[0]}^{d[n]} \frac{K}{d^2} \cdot \sin\theta$$

The resultant X and Y values represent the net force with which all objects detected by the LIDAR "push" on the car – this results in a net backwards Y force. To compensate, we imagine a huge positive charge located behind the positively charged vehicle, a gigantic, forward-pointing y-vector. We simply sum a large, positive constant into the Y to create this force. Since a forward-pointing vector has no left/right direction, it has no width. As such, no y-constant needs to be summed into the X value.

H. Week 3 Challenge

The culminating challenge for week 3 combined the tasks of obstacle avoidance and colored blob detection; an oblong arena with two "islands" (shaped vaguely like the greek character θ) was filled with various wooden boards and posts, with colored markers strewn throughout the arena, some taped to walls, others to obstacles. The task was to collect points; +10 points for each unique marker detected and catalogued, -2 points for each collision. Teams were given 2 minutes to explore the arena as thoroughly as possible.

Our team implemented a potential-field avoidance

system (after much struggling with radian-degree conversions) with the help of Winter Guerra[19], an MIT Associate Instructor. This potential field controller was paired with a blob detection system recycled from the week 2 challenge.

Video of our team's performance at the week 3 challenge can be found online:

https://youtu.be/SB-El-a54Pc

Our team performed highly, scoring third place overall with 4 successful detections and 0 collisions. Our team would have scored even more highly had our car not navigated into an area of low potential. In a T-intersection, a lack of walls in the X-direction resulted in no net turn for the vehicle. A very close distance to the wall ahead of the vehicle created a backwards repulsion force that equated the ever-present forwards vector. As a result, the vehicle was "pushed" equally in all directions and became stuck; a total of 20 seconds were spent exploring the environment. The remaining 1:40 were spent stuck in the potential well.

V. Week 4

The fourth week was spent in preparation for the final race challenge. Students implemented control, decision-making, and obstacle avoidance algorithms as they prepared to compete in a situationally and strategically complex final race. Successfully implementing <vision system> to toggle between potential field and wall follow controllers, our team placed <place> in the final race.

A. Race Format

The final challenge was deceptively simple; pass through a racecourse as quickly as possible. The race comprised of 3 rounds of runs:

1) Time Trials

During the time trial portion of the race, teams were given three runs to make it through the course as quickly as possible. The average of these three times would determine their placement in the heats.

2) Heats

Three ranks were formed; the fastest three teams, the middle three teams, and the slowest three teams. In the heat, three teams would have their cars lined up horizontally across the starting line (fastest car to the

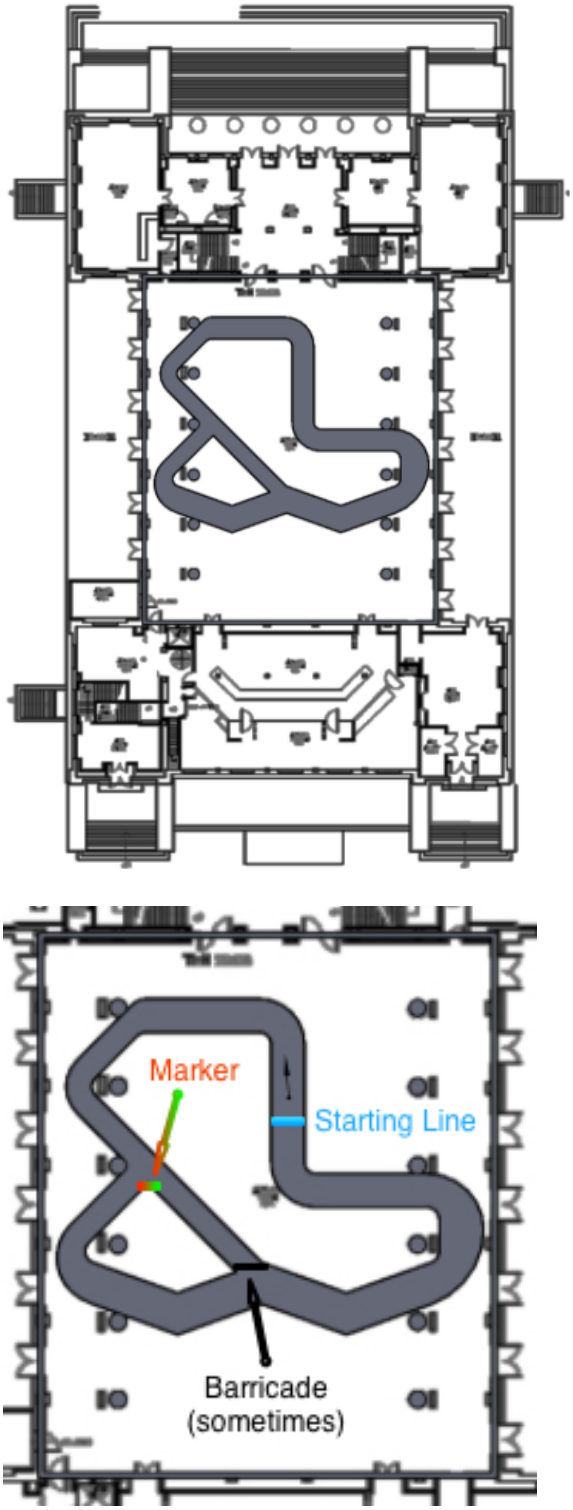

Fig. 4-1, 4-2. The final racetrack, laid out on the floor of the MIT Walker Memorial, Building 50.

inside of the track) and raced against the other two cars to determine placement in the Grand Prix.

3) Grand Prix

All 9 teams had their cars placed into the track at the same time; each heat was a row, lined up to form a 3x3 matrix behind the starting line. The winners of the individual heats were placed closest to the inside wall. The 3 fastest cars became the final champions.

B. The Shortcut

A shortcut would be open for only one of the time trials; a colored marker at the corner of the intersection would tell the car whether or not the path was open. To force competitors to use vision (and not simply rely on a wall-follow algorithm or LIDAR scans), a barricade was set up on the far end of the shortcut on the closed runs (thus, if the car failed to make the turn away from the shortcut it would be forced to make an extremely tight U-turn).

C. Obstacle Avoidance

In the heats and grand prix runs, the RACECAR teams were exposed to something they had not yet experienced in the course; moving obstacles. Having multiple cars in the same track created obstacles that the LIDAR unreliably detected; various points on the cars reflected the lasers back at the sensor, registering an obstacle, while holes in the vehicle allowed laser light to pass through, preventing the sensor from detecting an obstacle. This resulted in unpredictable behavior for nearly every car's controller, regardless of the algorithm, as both wall-follow and potential field controllers rely on accurate measurement of obstacles.

D. Developing Controls

The final racetrack was wide, nearly 6 feet across in many areas. This allowed a wide range of control algorithms to be feasible. Our team toyed with wall-follow, free-space, and potential field algorithms before developing a hybrid free-space/potential field system we implemented for the final race.

The wall-follow algorithm is simple and easy to execute. The wide track and lack of obstacles made wall-follow particularly appealing to our team, especially as it could allow us to closely follow the left wall (reducing the total distance our vehicle would have to travel). Following the left wall seemed like a good option until considering turns; a left-hand turn with a far-left wall-follow algorithm would force the vehicle to make a snap change in direction (the LIDAR data would read a left wall up until the turn, at which point it sees the wall suddenly disappear). This snap change could lead to oscillations in the PID control and an inefficient path.

Our team's second thought was then to implement a right-wall-follow system, following the right wall at a great distance (so as to still hug the left wall). A system such as this would allow the car to perceive left-hand turns more easily, hugging the greater, slower curvature of the right hand wall in order to avoid instabilities in the PID controller.

This system worked relatively well, but since both right and left hand turns existed in the racetrack (and so at least one snap change issue would exist regardless of which wall we followed), our team decided to explore other approaches.

We thought back to Oktay Arslan's advice in implementing a free-space system to pull the car simply in whatever direction had the farthest distance. Without obstacles in the course, this method would have worked well in the straightaways, but again, not well with turns; as soon as the LIDAR scanner passes into an intersection, the free-space vector would pull it with a tremendous amount of force towards the open pathway. Being that the LIDAR is mounted on the front of the vehicle, the rear wheels would clip the corner of the wall as it attempted to make any turn of significant magnitude. Being that this issue was unavoidable, our team sought yet another method of control.

We resurrected our potential field controller; this controller had not failed us in avoiding obstacles, so it ought not fail us in an open raceway. Indeed, the car drove with impressive stability, centering itself in the racetrack almost perfectly, and took shallow turns beautifully. The main issue we faced with the potential field system was making sharp turns, especially in the intersection with the colored marker. The potential field system is very good at producing stable control signals, but tends to make slow, wide turns and occasionally gets stuck in regions of low potential in intersections. To combat this issue, our team developed a hybrid system.

The combined free-space/potential field control system utilizes the strong, sudden pull vector produced by the free-space system on top of the signals produced by the robust potential field system. The RACECAR would calculate the resultant vector of the potential field and would sum that vector with a scaled free-space vector. To this end, the car would drive stably in open areas and when approaching a corner, would suddenly be jerked in the direction of the turn. The potential field system continued to produce repulsive

vectors from the walls, preventing the super-sharp turn produced by the vector alone. When tuned, this system was capable of taking sharp turns (<= 90°) at relatively high speeds (< 2m/s).

The system became unstable once the max speed was increased from 2m/s to 3m/s, but with slight tuning was incredibly successful.

E. Developing Vision

Developing vision in the Walker Memorial building was difficult for one reason in particular; as the day progressed, the sun would move through the sky and cast glare through the windows and orange overhangs. This resulted in the colored markers having different HSV values throughout the day as the light shown off of them differently. If the vision detector was tuned too specifically, it would cease to detect the colored markers correctly; if it was tuned to pick up a wide spectrum of marker-like colors, it would start detecting random items in the room. The constantly-shifting color scheme prevented our team from reliably making the shortcut decision and turn. This issue continued to haunt us through race day.

F. Making the Decision

Once our values were tuned correctly (at least for a given time of day), our vehicle perceived green and red and acted accordingly (green to drive straight, continuing through the shortcut, and red to drive right, through the standard circuit). Making the turn was difficult in itself; the free-space vector was not strong enough to make such a sharp turn, which required more than 90° in direction change. Once the vision system detected a red marker, it would create a huge, artificial, right-pulling vector that pulled the car rightwards for 1 second. If the system detected green, the potential field system was left active. Being that there is a gap in the wall at the right-turn in the intersection, a moderately-strong leftward-pulling vector was enacted for 1 second to keep the car moving forwards through the intersection.

G. Challenge Result Summary

Our racecar placed 8th in the first race and 6th in the final; being that our code had been tuned for afternoon colors, we had expected an early failure in the intersection. However, with a robust potential-field system, our car effectively avoided the other cars during the final race.

H. First Race

Our vehicle was the second vehicle on the track; the vehicle before us failed in the intersection, so we expected our code to also fail in the intersection. Being that every vehicle after the first two passed the intersection correctly, we assume that the lighting changed significantly during the course of the races and would have expected our code to work effectively later in the day.

Being that our vehicle did not pass the intersection correctly, we scored 8th place, having achieved a rapid time through the one successful run our vehicle made.

I. Final Race

Our vehicle placed 6th overall; with a robust potential field field system we effectively avoided the other cars as obstacles. Our vehicle had an early lead out of the starting line, dodging most of the other cars, but was hit during a turn by another team's vehicle (which we later learned had its safety controller deliberately disabled) and was thrown into the wrong direction. After a lengthy recovery, our vehicle turned back on course to complete the race in 6th place.

VI. Some Conclusions

The Beaverworks Institute experience was unlike any that currently exists; the education was executed beautifully and the people involved were truly inspiring. I, personally, grew a tremendous amount on both the social/interpersonal front as well as the technical front, and will certainly be leaving MIT to try and create a RACECAR experience at my own school.

A. Technical Conclusions

Any conclusions drawn from the Beaverworks technical design/lecture sessions must culminate with a tremendous respect for the complexity of autonomous systems currently in testing by Google, Tesla, and other commercial enterprises. LIDAR data did not allow even a vehicle as simple as ours to operate reliably in a controlled environment, let alone a dynamic environment. The collapse of our vision system after lighting changes and our inability to effectively calibrate a color detector for any time of day further signals a disconnect between what is theoretically possible and what is practically possible. The complexity of our own systems is dwarfed by the complexity of full-scale systems, inspiring a reverence for real-world engineering. Furthermore, being able to see the real-world applications of this technology through the professors, guest lecturers, and associate instructors made the education deep and tangible, allowing students to experience the potential of the

technology they are developing.

B. Personal Conclusions

Attending the Beaverworks program felt like coming home; the conversations that sprang up between students ranged from complex algorithms to philosophy, politics to practicality, in a way that only could exist among students with an avid curiosity towards the world. Furthermore, the staff and professors involved in the program were magnificent, treating the students almost like peers, rather than high schoolers, and did not hold back on depth of information during technical lectures. Living with students from across the country, students with very different backgrounds, but whose passion for exploration is unmatched, was a truly refreshing experience. My "home" community lacks students with this passion and makes communication with local peers difficult, if not impossible.

Participating in the Beaverworks program was an inspiration for me, personally, as it opened my eyes to a world that I did not realize even existed; intellectually brilliant and curious students, equally explorative professors, and taking a refreshing "drink from the firehose" was an unmatched experience and has helped re-form my goals as I prepare to apply to college. The experience of interacting with other students who are like me was novel and wonderful and has prepared me to seek out communities like that at MIT and has inspired me to try and build one back "home."

VII. Epilogue

In the weeks since I have returned to La Canada I have shared my experiences at Beaverworks with my local school district. The technology advisor for the district was thrilled with the possibility of having a RACECAR vehicle and curriculum at our local high school and is offering resources towards purchasing the hardware for a vehicle (and will be helping me fundraise to purchase multiple vehicles). A math teacher with some previous experience in computer programming has offered her classroom during a weekly homeroom period to host a small class of students whom we plan on teaching as much of the Beaverworks curriculum as we can.

I hope to emulate the community of students similar that I encountered at Beaverworks to the best of my ability; by creating an environment that attracts bright, intellectually curious students and gives them the resources to thrive, I hope to bring to La Canada a small piece of what I experienced at MIT.

On a more personal note, I have definitively developed as an individual from my experiences at Beaverworks. I have become content as an individual. I am looking forward, for the first time, in my life, to finding a community of students like me in university. I have seen that such a community can exist, and in fact thrives in places like MIT. I have new experiences to share and the inspiration and resources to build a community that, at least in part, seeks to bring a part of MIT's spirit and ideals to other students like me; students who live in despair, never realizing that people like them do exist in the world.

I hope to give these students a place to start looking.

References

1. "MIT Lincoln Laboratory Beaver Works Center." MIT Lincoln Laboratory, www.ll.mit.edu/news/beaverworkscollabs.html.
2. "RACECAR A Powerful Platform for Robotics Research and Teaching." *RACECAR A Powerful Platform for Robotics Research and Teaching*, Massachusetts Institute of Technology, fast.scripts.mit.edu/racecar/hardware/.
3. "RACECAR Hardware." *MIT RACECAR Github*, Github, github.com/mit-racecar/hardware.
4. "ROS Indigo Igloo." *ROS Wiki,* ROS.org, wiki.ros.org/indigo.
5. Edelberg, Kyle. "Introduction to Control Systems." MIT 35-225, July 2016, Cambridge, Massachusetts. Lecture.
6. "Bang-bang control." *Wikipedia*, en.wikipedia.org/wiki/Bang–bang_control
7. Liberzon, Daniel, and Stephan Trenn. *The bang-bang funnel controller: time delays and case study*. Coordinated Science Laboratory, U Illinois.
8. "PID for Dummies." *Control Solutions Minnesota,* www.csimn.com/CSI_pages/PIDforDummies.html.
9. "OpenCV." *Open Source Computer Vision*, opencv.org.
10. Georgieva, Lidiya, et al. *RGB and HSV colour models in colour identification of digital traumas images.*
11. "SimpleBlobDetector Class Reference." *OpenCV, Open Source Computer Vision*, docs.opencv.org/trunk/d0/d7a/classcv_1_1SimpleBlobDetector.html#gsc.tab=0.
12. Karman, Sertac. "Localization and Mapping: Dead Reckoning." MIT 33-116, Cambridge, Massachusetts. Lecture.
13. Callmer, Jonas. *Autonomous Localization in Unknown Environments.* Linköpings Universitet.
14. "gmapping." *ROS Wiki*, ROS.org, wiki.ros.org/gmapping.
15. Durrant-Whyte, Hugh, and Tim Bailey. *Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms.* Berkeley, U of California Berkeley.
16. Arslan, Oktay. Interview. July 2016.
17. "Coulomb's Law." *Farside, U of Texas*, U of Texas, farside.ph.utexas.edu/teaching/em/lectures/node28.html.
18. "Numpy." *NumPy*, Scipy.org, www.numpy.org.
19. Guerra, Winter. Interview. July 2016.