XE475 Final Project

1st Hassak
Department of Defense
United States Military Academy
West Point, New York
sam.hassak@westpoint.edu

2nd Cordray
Department of Defense
United States Military Academy
West Point, New York
matteo.cordray@westpoint.edu

***Abstract*—This report summarizes the design and results of a autonomous vehicle that can navigate find and engage targets without any human intervention. It also summarizes supporting theory around particular subsystems, such as Fuzzy Control, PID Control and line following object Detection. This project was extremely successful in achieving its specified goal of engaging targets when told to do so, and following a path.**

I. Introduction

The purpose of this lab is to design and build a system, using an MMP-5 Robot, an Arduino Mega 2560 rev3 development board, and Pixy2 cameras, that can navigate a path, select directions in the path based on external markers, identify multiple objectives within the environment, and execute appropriate actions based on the present objective. The system must be initiated by pressing a button, which should be able to start and stop the system at any time. The robot uses both Fuzzy control and PID control to maintain it's position for both line following and target applications, which work for both stationary and moving targets. Mechanical design for an appropriate shooting mechanism involved using a Servo motor along with rubber bands that rotated continuously around a screw.

II. Theory

A. Closed Loop Fuzzy Control

Fuzzy logic is a method of control that attempts to mimic human reasoning. It can be used to model complicated systems without the need for complicated mathematical models. Instead of the conventional model-based controllers, fuzzy logic uses expert knowledge, which is"knowledge possessed by human experts about a situation or problem." [1] A fuzzy system uses fuzzy reasoning to convert crisp inputs into crisp outputs. Fuzzy systems have three main processes: fuzzification, inference mechanism and defuzzification.

1) Fuzzification: As Lilly explains a fuzzy set "is a collection of real numbers having partial membership in the set." [1, ch 2] Unlike conventional, or crisp sets, a number can belong partially to one group and partially to another at the same time. Fuzzification is the process by which measured quantities from the process are converted into fuzzy sets to be used by the inference stage. A fuzzy set is defined by two things, "the members of the set and each member's degree of membership in the set." [1, ch 2]. The degree of membership is defined by membership functions. Although a membership function can be defined by any shape, two common types of membership functions are triangular membership functions and Gaussian membership functions. Both of them convey similar information: temperatures closer to 25 degrees Celsius are considered warm. [1, ch 2]

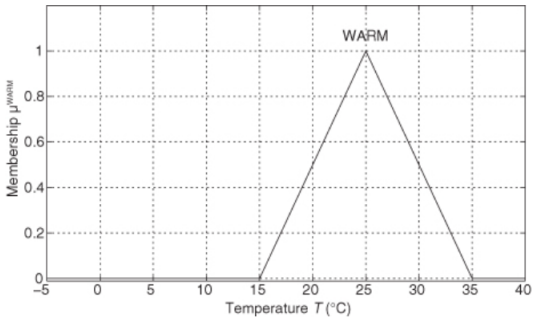

Fig. 1. Triangular membership function [1, fig 2.1]

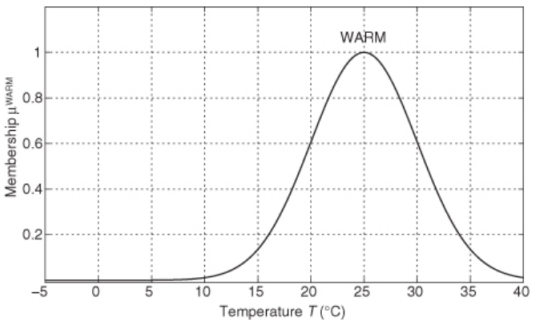

Fig. 2. Gaussian membership function [1, fig 2.2]

The only requirement of a membership function is that they make sense for the fuzzy set being defined. The excpetion is that the first and last membership functions are always one, that way there are no undefined outputs for a defined input. This is shown in figure 4. Figure 4 also demonstrates partitions of unity because the sum of all memberships is one at every input value. The fuzzy sets in figure 4 are also normal because their membership reaches one.

2) Inference: The second stage of a fuzzy controller is the inference mechanism. The inference mechanism determines the extent to which each rule in the rule base is applied to

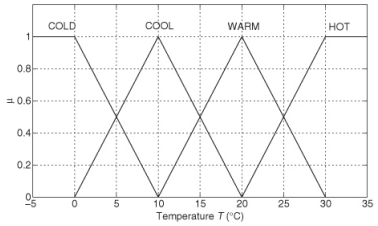

Fig. 3. fuzzy set of membership functions. [1, fig 3.2]

the present situation, and then forms a corresponding implied fuzzy set for each rule. The rule base is a series of if-then statements. [1] In this lab, a nested if statement was used to form a rule base of 25 rules based on both position and change in position.

3) Defuzzification: The final stage of a fuzzy controller is defuzzification. In the defuzzification stage a collection of degrees of firing of all rules are converted into a crisp output. In order to achieve a crisp output, all of the recommendations are combined by "taking a weighted average of the various recommendations." [1, ch 3.5] This can be done using several methods, but the one used in this lab was Center Average (CA) Defuzzification, so that is what this section will cover.

$$y^{crisp} = \frac{\sum_{i=1}^{R} q_i \mu_i(x)}{\sum_{i=1}^{R} \mu_i(x)} \tag{1}$$

If the consequent fuzzy set of rule i is Q^i and is characterized by the membership $\mu^{Q^i}(y)$. Then the crisp out of system will be given by 1. Note this method only works if consequent fuzzy set Q^i is normal, which in this lab it is.

B. Computer Vision

Computer Vision is the field of study within computer science that is centered around "enabling computers to identify and understand objects and people in images and videos." (pixycam) It works by using sensing devices and machine learning to attempt to replicate the way humans see. Computers recognize patters in visual data and use those patterns to determine what is present in an image. A brief overview of how computer vision works is presented below [2].

- A sensing device captures an image. The sensing device for this lab is the Pixy2 Camera.
- The image is then sent to an interpreting device. In this lab the interpreting device is also the Pixy2.
- The interpreting device uses pattern recognition to break down the image.
- The patterns identified are compared to a library of known patterns to determine if any known objects are detected.
- Useful information is then reported back to the user, such as the object detected, its location on the screen, its size, etc.

The drawback of computer vision is image sensors output a lot of data, and processing this large amount of data can be too much for many processors.

1) Pixy2 Camera: Pixy2's solution to image processing is to combine a camera with a dedicated processor. The sensor used in this lab is the Pixy2 Camera. It has the following specifications [3].

- Processor: NXP LPC4330, 204MHz, dual core
- Image sensor: 1296x976 resolution with integrated image flow processor
- Lens field-of-view: 60 degrees horizontal, 40 degrees vertical
- Power consumption: 140 mA typical
- Power input: USB input (5V) or unregulated input (6V to 10V)
- RAM: 264K bytes
- Available data outputs: UART serial, SPI, I2C, USB, digital, analog
- Dimensions: 1.5" x 1.65" x 0.6"
- Weight: 10 grams
- Integrated light source, approximately 20 lumens

It is capable of learning to detect objects at 60 frames per second with just the press of a button. It also has algorithms that can detect or track lines. The Pixy2 can be connected to a computer using USB, to an Arduino using a Serial Peripheral Interface, or I2C.

C. Actuators

An actuator is "a part of a device or machine that helps it to achieve physical movements by converting energy... into mechanical force." [**?**] Simply, it is a component of a machine that enables movement.

1) components: The following are usual components of a functioning actuator [**?**].

- actuator: This is the actual device that converts the supplied energy to a mechanical force, in this lab it was a DC motor.
- power source: This provides the energy necessary to move the actuator, in this lab the power source was a 11.4V lipo battery.
- power converter: The power converter takes power from the source and presents it in a manner that is within the specs of the actuator. For example, the battery might be 11.4V, but the motor likely takes less voltage than 11.4V.
- mechanical load: The energy converted by the actuator is being converted for a reason. The mechanical load for this actuator is the wheels that move the traxxis.
- controller: The controller allows the system to operate at the desired input. The controller for this lab was an arduino mega.

III. Shooting Mechanism Design

A. Hardware

The shooting mechanism consists of rubber bands stretched between a Servo and the tip of the "cannon."

1) Servo: We used a continuous servo because this allowed us to put as many rubber bands on the servo as it can hold. If we used a standard servo we would need to have a different geometry that could hold at least 4 rubber bands without making a full turn. For the servo we did select it would make a half rotation when turned at a defined constant speed for 600ms. This was determined experimentally.

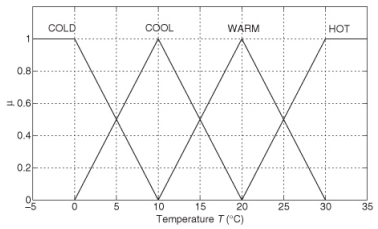

Fig. 4. fuzzy set of membership functions. [1, fig 3.2]

B. Software

1) Pixy2 Camera: The Pixy2 camera offered various ways to detect targets. In this lab, the color codes component was used to detect different targets: either enemy or friendly. We used PixyMon v2, a software that goes hand-in-hand with the Pixy2, to tune both the speed of recognizing targets and the lighting environment. Properly increasing the brightness, as well as increasing the sensitivity for the colors blue and purple, helped increase target recognition significantly. We decreased the color "yellow" in sensitivity because of the common items in the lab that the Pixy2 camera detected as "yellow", such as the wooden doors, desks, and chairs. The Pixy2's frame rate, blocks read per second, and all other parameters were not adjusted from the default values.

2) Arduino Mega 2560 rev3 Development Board: The Arduino Mega 2560 rev3, not to be confused with the Arduino IDE and the Arduino framework built on top of C++, was the microcontroller used to interpret data from the Pixy2 camera. Communication was conducted via I2C, a common communication protocol used for having multiple devices on the same shared bus, each having a unique "address". I2C was chosen for both Pixy2's because of it's simplistic setup and ease of use. Though SPI offers faster speeds compared to I2C, the difference would have been minimal because of the limitations in performance given by the Pixy2's processing power. The code used to properly setup I2C for the two Pixy2's to communicate with the microcontroller was given directly from the default Pixy2 libraries, using the popular "Wire" library for simple I2C communication.

3) Fuzzy Control library code walkthrough: In order to simplify code by reuse, a C++ header file was created for Fuzzy Control. Functions are broken up to provide filtering, updating membership functions, setting singletons and midpoints, as well as getting a crisp output.

Filtering used an Exponential Moving Average Filter (EMA) in order to properly filter the data for any variable coming from the Pixy2, including both the height and x position of the target. The library defines this as a "variable", and an EMA filter is used to turn this to a "variable" average, which is used throughout the rest of the library.

```
void filterPixy(int newData){

  // init variables
  int variable_sample = 0;
  int variable_v_sample = 0;
  variable_avg = 0.0;
  variable_v_avg = 0.0;

  // filter data
  variable_sample = newData;

  // calc variable average
  variable_avg = (variable_sample * _alpha)
  +
  ((last_variable)*(1-_alpha));

  // update time
  variable_v_time = millis();

  // update sample
  variable_v_sample =
  (
   (variable_avg - last_variable)*100.0)
   /
   (variable_v_time - last_variable_v_time
  );

  // update average
  variable_v_avg =
   (variable_v_sample * _alpha_v)
   +
   ((last_variable_v)*(1- _alpha_v));


  // update last variables
  last_variable = variable_avg;
  last_variable_v = variable_v_avg;
  last_variable_v_time = variable_v_time;

}
```

Listing 1. Fuzzy Control Library - Filter

Updating membership for the fuzzy inputs involved using "tmf", "tmfr" and "tmfl" functions, both internal to the library, to determine where membership lied for a certain input variable.

```
void updateMembership(){

    // update position variable membership
    memVariable[0] = tmfl(
                             variable_avg,
                             midVariable[0],
                             midVariable[1]
                         );
    memVariable[1] = tmf(
                             variable_avg,
                             midVariable[0],
                             midVariable[1],
                             midVariable[2]
                        );
    /*
    ...
    trend continues...
    ...
    */
    memVariable[4] = tmfr(
                             variable_avg,
                             midVariable[3],
```

```
                              midVariable[4]
                            );

    // update velocity variable membership
    memVariableChange[0] = tmfl(
                                      variable_v_avg,
    midVariableChange[0], midVariableChange[1]
                                 );
    memVariableChange[1] = tmf(
                               variable_v_avg,
                               midVariableChange[0];
                               midVariableChange[1],
                               midVariableChange[2]
                              );
    /*
    ...
    trend continues...
    ...
    */

    memVariableChange[4] = tmfr(
                               variable_v_avg,
                               midVariableChange[3],
                               midVariableChange[4]
                               );

    }
```

Listing 2. Update membership

```
int inference(
 float (*two_d_array)[5][5],
 float (*mem_two_d_array)[5],
 float (*mem_vel_two_d_array)[5],
 int (*rule_matrix)[5][5]){

  float numerator = 0.0;
  float denominator = 0.0;
  for(int i = 0; i < 5; i++){
    for(int j = 0; j < 5; j++){

      (*two_d_array)[i][j] =
        (*mem_two_d_array)[i] *
        (*mem_vel_two_d_array)[j];

      numerator +=
      (*two_d_array)[i][j] * (*rule_matrix)[i][j];

      denominator += (*two_d_array)[i][j];

    }
  }

  return (int) (numerator / denominator);

}
```

Listing 3. Crisp Output Inference function

Singletons provide a means of receiving a stable, crisp output. We needed to establish a function that could set singletons up the way we needed them too.

```
void setSingletons(int FF, int FS, int Z, int BS,
    int BF){
      _FF = FF;
      _FS = FS;
      _Z = Z;
      _BS = BS;
      _BF = BF;

      // Input sets
      int newRule[5][5] = {
        {_FF, _FF, _FF, _FS, _Z},
        {_FF, _FF, _FS, _Z, _BS},
        {_FF, _FS, _Z, _BS, _BF},
        {_FS, _Z, _BS, _BF, _BF},
        {_Z, _BS, _BF, _BF, _BF}
      };

      for(int i = 0; i < 5; i++){
        for(int j = 0; j < 5; j++){
          rule[i][j] = newRule[i][j];
        }
      }

      (*p_rule)[5][5] = &rule;

    }
```

The only issue with this setup, is that the rule set is static, meaning that we could not adjust individual rule sets or size, we could only change the value of each singleton (FF, FS, Z, BS, and BF). Pointers were used to use less memory and have a faster response for the microcontroller.

In order to receive a true, crisp output, we created an inference function that assigns degrees to each rule in the rule base for height and x positions.

```
// since in c++ you cannot pass a double array,
// pass a pointer to the array
```

4) Searching, finding, and engaging with target: In order for the MMP-5 to find, detect, and engage with the target, a searching and aiming function was used to properly deal with the objective. The MMP-5 simply rotated about it's main center of mass until a color code was detected. Any color code's height that was less than a value of "10" was ignored, since no targets were that far away from the objective marker and allowed us to filter out any targets that are present on the course but not designated to the current objective.

```
bool biggerBlock = false;

// while there are no blocks on the screen...
while(!biggerBlock){

 pixy_cannon.ccc.getBlocks();

 if(pixy_cannon.ccc.numBlocks > 0){
   if(pixy_cannon.ccc.blocks[0].m_height > 10){
     biggerBlock = true;
   }
 }

// Turn MMP5 to the right
right_motors.write(1350);
left_motors.write(1550);

}
```

Listing 4. Searching function

In order to engage with a target, whether a friendly or enemy, we set the MMP-5 to decide what target it is, and use Fuzzy control to center it's position with the color code.

In the case of the enemy, speed is of the essence. We determined that since the target is rather large (8.5" x 11") and not far from the lined-track, accuracy does not need to be incredibly precise, just enough to knock the target over. Which, as covered in the hardware design, the rubber bands provide a perfect amount of strength to knock over a target no matter

where it hits on the target. Therefore, we added a ”deadband” zone and a counter. When the ”x” position returned by the Pixy2 is within this ”deadband” zone, for enough time, we know the target can be engaged with easily.

```
if(counter > 0){
      if( // start if
       steering <= (STOP + DEADBAND)
       &&
       steering >= (STOP - DEADBAND)
        ){ // end if
        counter++;
        // if the counter is above 10 values
        if(counter >= 10){
          counter = 0;
          centered = true;
        }
      } else {
        // if the previous value is not within range
        // reset the counter. This is because we are
        // looking for ten CONSECUTIVE values
        counter = 0;
        centered = false;
      }
    } else if (counter == 0){
      if( // start if
       steering <= (STOP + DEADBAND)
       &&
       steering >= (STOP - DEADBAND)
        ){ // end if
        counter++;
      }
    }
```

Listing 5. Counter and Deadband code

Moving targets, such as the boss, require the algorithm to shoot while still adjusting to the center of the target. This is because the time it takes for the shooting mechanism to begin firing and release a rubber band are not instantaneous. There is a small delay between the time the servo starts to rotate and a rubber band is released. Therefore, our code includes a non-blocking timer, which lets the program continue to adjust the target's position on the Pixy2 via Fuzzy control, while also engaging with the target. To engage with the target, as stated in the hardware section, the servo moves a half rotation to allow the next rubber band to be ready to fire for the next target.

```
if(centered){

    if(!ranOnce){
      cannon.attach(CANNON_PIN);

      // Turn servo (allow rubber band to fire)
      cannon.write(1800);
      previousMillis = millis();
      ranOnce = true;
    }

    unsigned long currentMillis = millis();
    if(currentMillis - previousMillis >= 600){

      // Stop servo from moving
      cannon.write(1515);
      cannon.detach();

      return true; // CHANGE TO FALSE WHEN TESTING
    }
    return false;
} else {
  return false;
}
```

Listing 6. Non-blocking timer and Boolean logic for determining center

The case for the friendly target is almost identical, with the only adjustment being for throttle control using the Fuzzy library. The same counter is used, but includes both steering fuzzy output sets as well as throttle fuzzy output sets lying within the deadband region.

IV. Line Following

A. Control

The Line Following mechanism used PID control for its ability to maintain and follow a line throughout the course. Though it used a PID controller, Proportional Control was enough to satisfy the needs of the lab, since the main advantage for the MMP-5 is the differential drive features to provide both steering and throttle.

The controller was first tuned by starting with a proportional gain of 1. While it provided solid performance, the response time was too slow on sharper turns versus a higher proportional gain. After slowly increasing the proportional gain, we reached a value of 10 that satisfied the requirements needed for the project.

B. Hardware

Our robot started with a MMP5 as the base. Screwed onto the base was a two layer base plate. A bottom layer made of plastic and a top layer made of sheet metal. Below the top plate is where we housed our arduino, and a breadboard. Above the plate to the rear is a mount for the pixycam that searches for targets. To the rear of that mount is a spare battery to power the second Pixy2. In front of the first Pixy 2 mount is an extended arm. At the base of this arm is a servo that is used to hold the projectiles which are rubber bands. At the tip of the arm is another Pixy2 which is used to conduct line following and barcode reading. These two Pixy2 cameras are connected to the arduino using I2C.

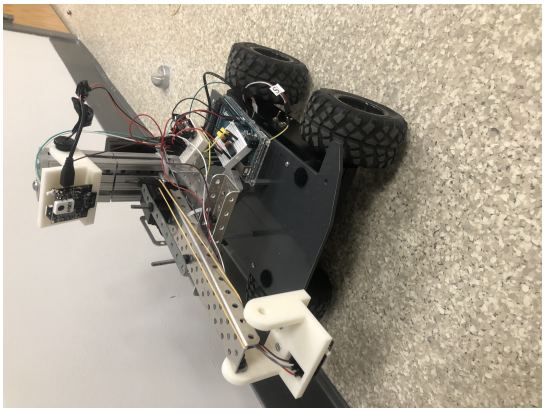

Fig. 5. Our Robot

C. Software

Error was relatively easy to calculate, since all we needed was the x position of the vector (head) and the center of the Pixy2 camera's view.

```
// calculate the error
error_line =
    (int32_t) pixy_line.line.vectors->m_x1
    -
    (int32_t) X_CENTER;

// Perform PID calculations
Direction.update(error_line);
```

Error is calculated using the PID header created to make PID easily readable in the main loop function.

```
void update(int error){
  int32_t pid;

  // if previous error is not undefined
  if (_prev_Error != 0x80000000L){
    pid = (error * _propGain);

    // Deal with deadband
    if (pid>0){
      pid += DEADBAND;
    } else if (pid<0) {
      pid -= DEADBAND;
    }

    motor_signal = (int32_t) pid;
  }
            _prev_Error = error; // Update memory
    for derivative term
}
```

The Pixy2 camera could detect lines as a vector, indicating both a tail and a head. The tail and head gives us both direction and orientation. We know that if the head y position is greater than the tail position, the vector is pointing away the MMP-5, which gives way to adjust the center of the robot appropriately. In order for there to be time for the MMP-5 to make a decision on an intersection appropriately, we slow down when the Pixy2 sees an intersection is present in the field of view.

```
// If vector is pointing away from us...
if(
    pixy_line.line.vectors
    ->
    m_y0 > pixy_line.line.vectors->m_y1
  ){

    // Slow down if intersection is present so the
    pixy has enough time to register the event
    if(
     pixy_line.line.vectors
     ->
     m_flags&LINE_FLAG_INTERSECTION_PRESENT
      ){

      left_dir += SLOW;
      right_dir += SLOW;

    } else { // if no intersection is present, keep
    going
      left_dir += FAST;
      right_dir += FAST;
    }
}
```

V. System Integration and Testing

For the system integration, we used a variety of components to complete the project. Mainly, a voltage regulator to give current to the Arduino, Cannon servo, and a Pixy2, and a Buzzer. In addition, a AA battery pack provides power for the Cannon mounted Pixy2 camera.

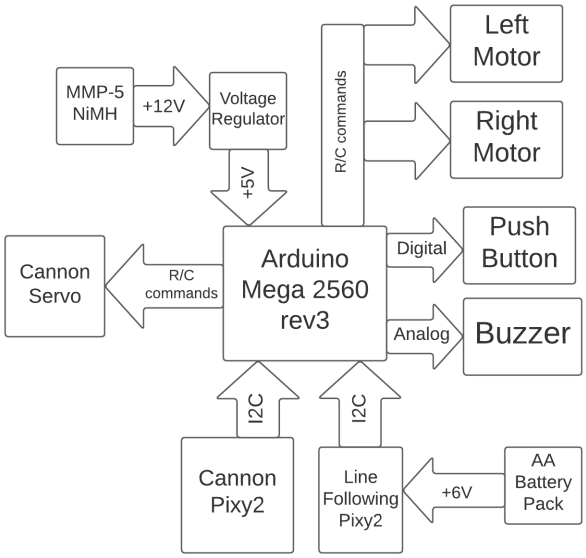

Fig. 6. Block Diagram of Entire System

A. Software

For all the sub pieces, the code is setup in a way to perform a continuous loop. Our main loop checks for line following first, as the main event in the course is following a line. The loop includes receiving the main features, which range from lines to barcodes, and addresses each event accordingly after receiving updates from the Pixy2.

```
void loop(){
  // Get latest data from Line Following Pixy
  mainF = pixy_line.line.getMainFeatures();

  if(mainF <= 0){
    Serial.println("no line found");
  }

  if(mainF&LINE_VECTOR){
    followLine();
  }

  if(mainF&LINE_INTERSECTION){
    printIntersection();
  }

  if(mainF&LINE_BARCODE){
    if(barcode == STOP){
        while(1){}; // done with course
    } else if (barcode == RIGHT){
        setNextTurn(-60);
    } else if (barcode == LEFT){
        setNextTurn(60);
    } else if (barcode == OBJECTIVE){
        searchForObjective();
        if(enemy){
```

```
            rightOrLeftSide();
          } else if (friendly){
            moveBackwardsUntilLineFound();
          }
      } else if (barcode == BOSS){
          searchForObjective();
      }
    }
}
```

Listing 7. Main loop semi-pseudocode

In order to adequately search for and find the objective, the robot scans the area by turning right and finding a color code, which brings it to sequentially decide whether the color code is an enemy or friendly. We do not care if the color code is a boss, because we know that at the end of the course there will only be one target at the end of the course. In addition, we are reusing the same "shootEnemy" function with the boss that we used previously with the enemy targets.

```
void searchForObjective(){
  pixy_cannon.ccc.getBlocks();
  while(blocksNotFound == TRUE){
    pixy_cannon.ccc.getBlocks();
    turnRight();
  }

  // if the block is equal to code "1-2"
  // signature received from Pixy2 is 11, not 12...

  if (block == 11){ // Friendly target
    goToFriendly();

  } else if (block == 10){ // Enemy target
    shootEnemy();
  } else {
    shootEnemy(); // if boss is encountered...
  }
}
```

Listing 8. Search for objective semi-pseudocode

The *goToFriendly();* function serves to approach the friendly, play a tone, and move back towards the line. In order to accomplish this task, this function continues the state-machine by going through the sequential steps needed to communicate with a friendly unit and go back to line following.

```
void goToFriendly(){

  // do nothing if target is not centered
  while(!centerOnTarget()){};

  // play tone
  tone(BUZZER_PIN, 3000);
  delay(1000);
  noTone(BUZZER_PIN);
  delay(1000);
}
```

This loop continues forever until either the stop code is reached or the end of the line is reached. If it reaches a stop code, the program is terminated and must be reset in order to restart the course.

B. Testing

Testing is a crucial part to making sure the system does not fail when presented for a proper demonstration. Therefore, the goal for testing is to complete each portion is different segments. This includes the following:

1) Complete line following mechanism and Proportional Controller
2) Successfully identify barcodes and distinguish between all the ones available (right, left, objective, stop)
3) Successfully switch between line following Pixy2 and cannon Pixy2
4) Identify whether target is friendly or enemy
5) Use fuzzy control to center on target continuously
6) Implement non-blocking shooting mechanism code
7) Implement fuzzy control for both throttle and steering (friendly)
8) Figure out if target is on left or right side (enemy)
9) Constantly track boss and hit target

This testing will ensure that project goals are met and the right steps are being taken to address any issues that come up during the project.

VI. Results and Discussion

A. Results - Shooting Mechanism

1) Accuracy, Precision, and Range: The accuracy, precision, and range all affect the completion of acquiring and hitting a target. However, multiple factors came into play when addressing these variables. The Pixy2's range was a limiting factor to the shooting mechanism's ability to hit a target. As distance from the target increased, the PixyMon software connected to the Pixy2 showed more noise and less recognition of the color code.

In terms of accuracy, our shooting mechanism was highly effective at close ranges, hitting the target about 90% of the time when tested at three different ranges: 1ft, 3ft, 6ft. At 1ft, the target was hit 100% of the time after 5 trials. At 3ft, the target was hit 80% of the time after 5 trials. Finally, at 6ft, the target was hit a total of 2 times after 5 trials.

In terms of precision, the results varied. Precision was captured by using a high-speed camera (iPhone X) and recording a video in slow-motion to capture where the target was hit by the rubber band. Using this, a piece of tape was placed on each spot shot by the shooting mechanism. Overall, precision was not as good as accuracy, but our goal was still attained by hitting the majority of targets, by a long shot. Our Fuzzy controller proved to be successful by successfully

Fig. 7. Rubberband hitting the target

engaging targets and knocking them down. However, their true performance proved to be somewhat opposite. The x position

on the Pixy2 camera had a 10.2564% overshoot, and the move value had a 10.3448% overshoot, and a settling time of 1.9280 seconds until firing the shooting mechanism.

Fig. 8. Enemy's position of the Pixy2 Camera, demonstrating the response time and overshoot for both the x position and the signal sent to the MMP-5

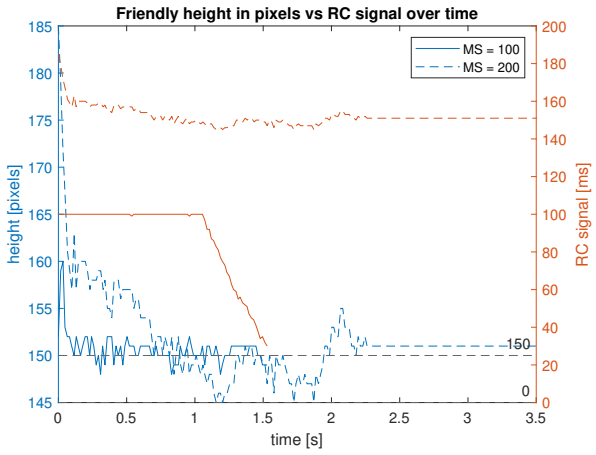

Fig. 9. Friendly position of the Pixy2 Camera, demonstrating the response time and overshoot for both the height and the signal sent to the MMP-5

B. Results - Line Following

Using proportional only control for the line-following proved to be successful, with little overshoot and a fast response time. Specifically, the robot would hug the line very tightly during turns as a result of the high gain, but it would occasionally oscillate on straight lines. This indicates that the gain is likely too high causing an underdamped system.

C. Results - Target Recognition and Engagement

During our actual test our robot never failed to recognize a target. During actual testing because of the angle on the camera after 1.44 meters the target would no longer be in view of the camera. This could potentially be fixed by aiming the camera higher, but this was not an issue for our use case.

Figure 10 shows the target acquisition of an enemy. Although there appears to be noise in both the signal and x position of the enemy, this noise is a result of our Pixy2 sensor changing the box around the detected object. There was little to no visible overshoot in the actual response.

VII. Conclusions and Future Work

A. Significant Results

The most significant result from this lab is its success. Our robot met all of its objectives, and performed more consistently when compared to other robots in our class. Using the MMP5 instead of a traxxis essentially allowed our entire robot to be the turret instead of mounting a turret on a servo on top of a traxxis. This extra stability allowed us to make a simple turret that would perform consistently.

B. Suggestions

Our group came up with two key suggestions. First the class should implement more mandatory 3D printing. Several lessons were devoted to 3D modeling, which seems like a useful skill, but it was then never used again. Secondly, We believe every group should learn to operate a differential steering robot. The MMP5 was not much more difficult than the Traxxis robot to operate, but it was significantly more effective than the Traxxis.

C. Future Work

In the future, we desire to create a 3D printed enclosure for more precision and accuracy on the rubber band "cannon". A 3D printed enclosure will not only look better, but also function better, as wires will not have the ability to be separated as easily and the shooting mechanism will be sturdier. A custom PCB was desired, but time did not allow for the completion of this. In the future, we wish to add a custom circuit board to make connections easier to troubleshoot and manipulate.

REFERENCES

[1] J. H. Lilly, *Fuzzy Control and Identification Lilly/Fuzzy Control*. Hoboken, Nj, Usa John Wiley & Sons, Inc, 2010.

[2] "What is computer vision? — microsoft azure," azure.microsoft.com. [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-computer-vision/

[3] "wiki:v2:overview [documentation]," docs.pixycam.com. [Online]. Available: https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:overview