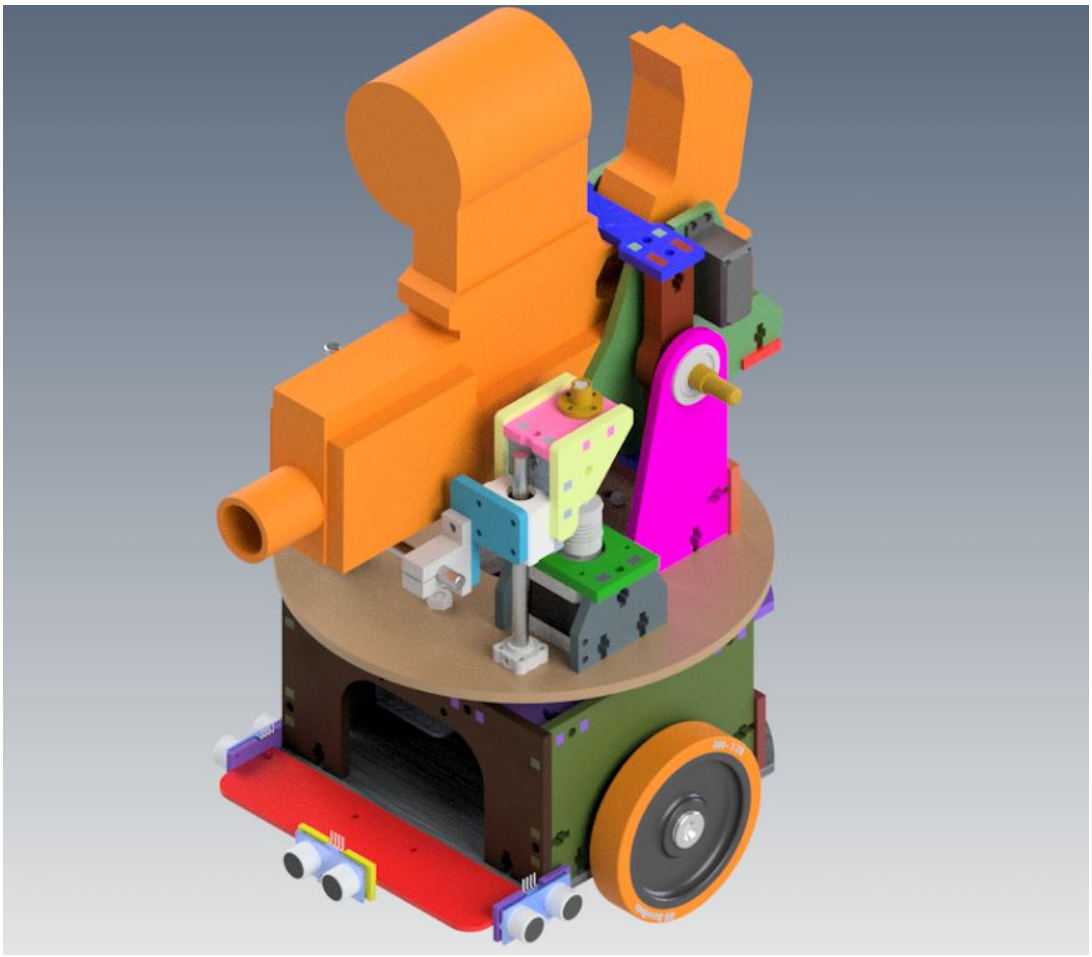


MCEN 5115: Mechatronics and Robotics
Police Academy Project Final Report

Jack Center, Chadd Miller, Arpit Savarkar, Scott Scheraga
CU Boulder, April 30, 2020

Abstract

The march towards next generation mobile robotic and targeting systems inspired the concept for the Police Academy Robot. Many surveillance applications are limited by human capabilities and safety. The risk and burden placed on human operators can be significantly reduced by further developing autonomous systems to better localize and track objects of interest, and then make prudent decisions with the information available.

The purpose of this project was to design an autonomous vehicle capable of completing a Police Academy style shooting range. The project goals were to navigate the hallways along a known map, locate and differentiate red targets from green bystanders, then knock down the targets with a projectile. The major design constraints for the project included a $200 budget, hallways that were 14" at the narrowest point, an 18" bridge that the robot would need to fit under, and targets which could be located at various heights. Additionally, some common sense safety considerations needed to be made in terms of the projectile used since innocent human bystanders would be within range of the projectile launcher.

The stand-alone robotic entity senses, perceives, localizes, tracks and path plans to find criminal targets on a known map. The Twist/Effort torque required to propel generate the friction for driving the robot is produced by two high geared encoded torque motors attached to wheels. The robot's odometry and pose areis kept track of by analysing the encoders of left and right wheels, perception and environment sensing is done using Ultrasonic Sensors. Targets are detected and distinguished by a depth camera and the turret azimuth and angle is set by two stepper motors. The system is able to navigate the course, visually acquire valid targets, position the turret, and fire darts. Future work will include integrating the target acquisition system with the turret mechanism to allow for autonomous positioning of the turret.

System Architecture

Architecture is a representation of entities organized in a way that supports reasoning about the entities and describes behaviors and relationships amongst the entities. Based on the analysis of form and function, the architecture system breakdown analysis is undertaken in the provided table.

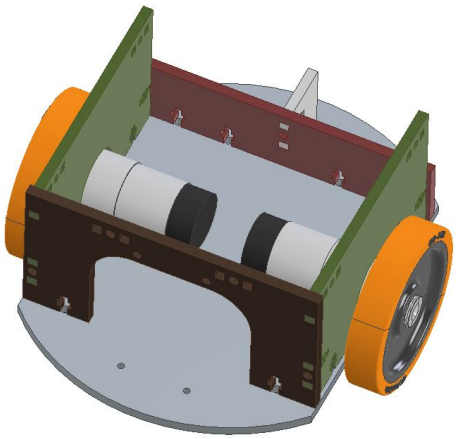
Architectural decision	Rationale	Option(s)
SBC (Single Board Computers)	Robotic System requires a powerful independent Processing Unit	Raspberry Pi
Actuator	Driving System with necessary torque for movement over ground	Motors with Quadrature Encoders
Perception Sensor	Ability to Localize is vital for robot	Camera and Ultrasonic sensors
Display - Monitor	Ability to view and track status of robot	Analog Instruments
Power Source	A power source to drive the robot autonomously	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
Controller	A mechanism to Override to Manual Mode in case of breakdown	Force Shutdown, Joystick, or Keyboard Control
Firmware / Hardware	Signal Processing, and Voltage Control	ADC, DAC, Motor Driver, MOSFET and regulators
Chassis	A body to mount peripherals and actuators	3D Printed frame, Body Frame-Steel, Brackets and molds
Peripheral Protocol	Wires and communication protocol between modules	I2C x12, ADC x5, 5pin OLLO x4

Form and Functional breakdown:
The functional breakdown is based on how coupled and/or sensitive the data in question is. Architecture decisions to reason about why the specific rationale was undertaken is broadly classified based on sensitivity and coupling. Sensitivity is a measure of the impact on metrics caused by a given decision. Connectivity is the degree to which making a given decision influences other decisions. For example choosing over communication between i2c vs Serial impacts the future coding and sensor selection. Additionally choosing over which Single Board Computer to use would impact the processing capabilities of the robot. Changing the display monitor/screen does not affect any sub-system but is an essential part to visualize and operate the robot. The table describes the Sensitivity vs Coupling for our Police Academy Robot. **The Link to the mechanical model of the robot is in the Appendix.**

	Loosely Coupled	Highly Coupled
High Sensitivity	Sensors Remote Control and Protocol	Single Board Computer(Processing Power) Power Source – Battery Chassis
Low Sensitivity	Display Monitor	Protocol Actuators Firmware

Chassis

The chassis is the first of the two major sub-components of the design. Propulsion is handled by two motors with two casters for balance. Major components include: one Arduino UNO, two DC motors with encoders, and three ultrasonic sensors. Initial plans included using a PixyCam for line and intersection detection, however, it was removed for simplicity, space, and due to the ultrasonic sensors being the focus for navigation. Overall, the chassis receives acceleration commands from the Raspberry Pi based on measurements from both the encoders and the ultrasonic sensors and attempts to remain equidistant from the walls on the course, with minimal heading change, while moving forward a specific distance. The chassis frame was designed with a number of goals: low cost, ease of manufacture, and ease of repair/component access. To those ends, the team decided to lasercut the frame out of .25" thick acrylic plates, and locked together with the Captured fastener assembly method.



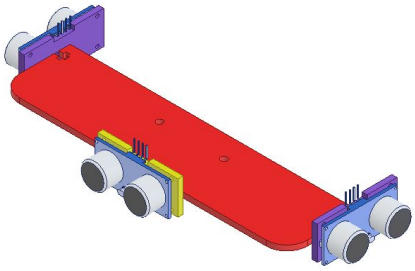
Chassis Actuators

The motors used were two Pololu "131:1 Metal Gearmotor 37Dx73L mm 12V with 64 CPR Encoders" and were driven by the Arduino UNO with PWM commands through a SparkFun "Dual TB6612FNG Motor Driver". The Arduino code is available under the base folder in the GitHub repository which makes use of the SparkFun motor driver Arduino Library. This library allowed for simple commands to drive the motors in either direction at a specific PWM, or to quickly stop the motors.

Commands for the actuators are based on translation and rotation. Given an input translation command, the two motors' PWM values are changed uniformly to the newly commanded values. When a rotation command is received, the motors' PWM values are changed in opposite directions at approximately equal magnitudes. The commands for translation and rotation are independent, but are combined into a single change in PWM for each motor. This allows the chassis to simultaneously drive and turn.

Chassis Sensors

The chassis makes exclusive use of the encoders and the ultrasonic sensors. Encoder values are read on the Arduino UNO, but not used on the microcontroller directly. Instead the data is sent to the Raspberry Pi for processing. The three ultrasonic sensors are located on the front of the chassis. One points forward and is used only in the localization algorithm. The other two ultrasonic sensors are located on either side of the chassis, one on the left and one on the right, just in front of the wheels. These sensors are initially used for localization, then, when executing a motion plan, are the primary sensor for determining how far the chassis is from the center line.



Control

Control of the chassis was implemented based on a five-state differential drive robot:

$$x(t) = \begin{bmatrix} x \\ y \\ \theta \\ \omega_l \\ \omega_r \end{bmatrix} \qquad u(t) = \begin{bmatrix} \dot{\omega}_l \\ \dot{\omega}_r \end{bmatrix}$$

$$\begin{aligned} \dot{x} &= r\,u_\omega \cos(\theta) \\ \dot{y} &= r\,u_\omega \sin(\theta) \\ \dot{\theta} &= \frac{r}{L} u_\psi \\ \dot{\omega}_l &= u_l \\ \dot{\omega}_r &= u_r \end{aligned} \qquad \begin{aligned} u_\omega &= \frac{u_l + u_r}{2} \\ u_\psi &= u_r - u_l \end{aligned}$$

Here, the state is represented by relative position coordinates and the orientation and the angular velocity of the wheels. The inputs u_omega represents translation and u_psi represents rotation.

For simplicity, the values for x, y were considered to be relative to the chassis' frame. Using the small angle approximation, the value for x was forward and backward, regardless of the orientation and the y value was considered the distance the robot was off of the centerline. However, the value for the sin of theta was taken as zero, so all motion was considered to be in the x direction. Due to the design of the map, the robot would only need to move along cardinal directions (north, east, south, or west) making this reference frame possible. This model allowed for the robot's motion to be controlled in a simple, although non-intuitive, way. Instead of simply commanding the chassis to a specific global state on the map, a series of locally oriented state commands would need to be given.

Each state was controlled through an individual Proportional Integral Derivative (PID) control law based off of three separate modes: forward, backward, and turn. Each of these modes required different coefficients for the PID controller to achieve the desired performance. The x value was used as the reference the system was being driven to in both forward and backward mode. To get the chassis to move forward 20 inches, the reference state for x was set to 20 more than the current state, and the controller would send commands to drive the robot to the newly commanded state. To move backward, the reference state would be set to a value less than the current local state. Encoder values were mapped to distance through the following formula:

$$C = \frac{2\pi r}{cpr} = \frac{2\pi(1.95'')}{100} = 0.1225\,\frac{in}{count}$$

Here, r is the radius of the wheels and cpr is the counts per revolution the encoders read, which was set to 100. The value for C, the conversion from encoder readings to distance, was initially set to 0.1194 inches per encoder count based on the physical measurements of the radius of the wheels and axle length being 1.9 and 8.25 inches, respectively . After experimentation, the final value for C was set to 0.1225 which provided the most accurate distance matching.

The reference for y was always set to zero so the controller would always attempt to keep the robot in the center of the lane. In order to avoid the controller needing to use walls that were not immediately adjacent to the chassis, all ultrasonic readings of greater than 20 inches were ignored and the value was instead set to four inches, which is how far the robot would be from the walls when traveling down a wide hallway. However, this meant there were three different scenarios that the ultrasonic sensors could be in that needed to be accounted for: both sensors, only one sensor, or neither sensor are getting appropriate readings.

The first case was both the left and right sensors were both getting readings of less than 20 inches. In this case, the difference of the two sensors was taken and the controller worked to drive the value to zero by turning the chassis. In the second case, if, for example, the left sensor was picking up an erroneous reading of 400 inches due to there being a long hall on the left of the chassis, its value would be set to four. The controller would then take the difference of four and the reading from the right ultrasonic sensor and drive the value to zero. This would have the effect of keeping the chassis four inches off of the wall on the right. In the final case, where the robot is in a four way intersection and both values are set to four, the controller assumes it is in the center of the lane because four minus four is zero, which is the reference value. In this case, the robot relies entirely on the encoder values and dead reckoning to proceed until another wall comes into range. Additionally, it is worth noting that this functionality was not used when the robot was driving backwards due to the inverse nature of the measurements to control inputs. In this case, the robot operated solely on encoder measurements and is why the chassis was not driven long distances in reverse.

The value for theta is maintained through the difference in encoder values. When driving forward or backward, the difference is driven to zero so the chassis maintains a constant heading. When turning, the controller drives the chassis to a new reference based on setting a difference in encoder values based on:

$$D = \frac{180}{L\pi}C = \frac{180}{8.25\pi}C = 0.8509\frac{degrees}{count}$$

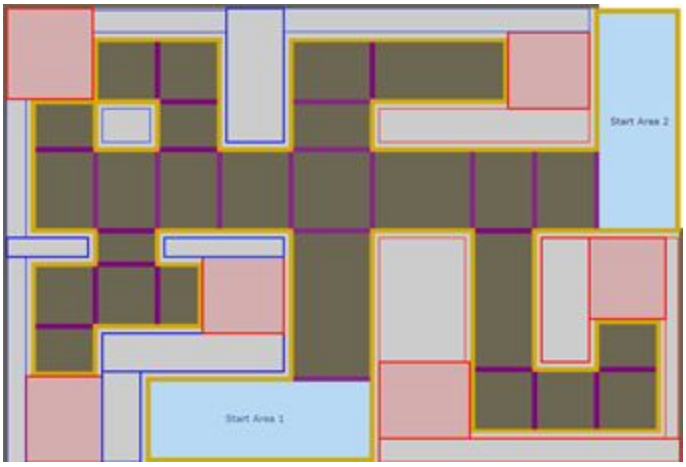
Here, L represents the length of the axle between the two wheels and The value for D was set to 0.8509.

The final two states were the angular velocities of the motors which were derived from the encoder readings. These states were used to limit the speed of the chassis under different modes of travel. The controller would only command inputs based on these states if the chassis was driven above the reference. That is, commands from this PID would only reduce the absolute speed of the motors, never increase it, and would only do so if the speed of the motors broke a certain threshold.

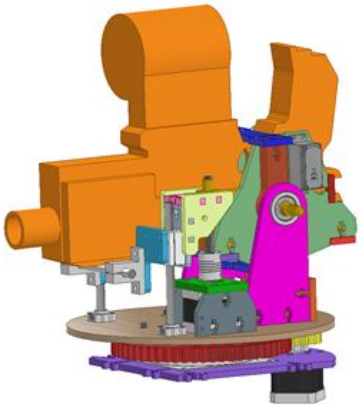
The commands from all five PID controllers were summed together and limited to range from negative three to thee. This was implemented to ensure smooth chassis motion and avoid wheel slip and protect the robot from high magnitude commands causing unexpected behavior. This command for translation and rotation is then sent to the Arduino UNO for implementation.

Decision Making

Localization was to be implemented based on the unique ultrasonic measurements available at the two possible starting locations. The basic algorithm was for the chassis to find the only corner in the starting area, turn towards the far wall, then drive forward until it was four inches from the far wall, then turn in the direction of the last wall it was following. This would allow for the two distinct starting positions to be identified and the chassis to be able to implement a predetermined motion plan based on the information.



Once the robot is localized, it executes a series of motion plans by deliberately changing the PID reference to achieve the desired motion. For example: forward to 56 inches, turn 90 degrees, forward to 78 inches, backward to 41 inches. Because the map was known, the motion planning was completed offline so all the robot needed to do was execute the maneuvers. It was implemented via a text document where each line is a separate command, examples for which are in the GitHub repository under the "cmds" folder. This method of updating the motion plan struck a balance between ease of use and simplicity of implementation. At specific way points referred to as firing points, the chassis would stop executing the motion plan while the turret implemented the search and destroy sequence. Once the sequence completed, the robot would continue to follow the motion plan until the next firing point. Links to a simulation of the robot executing a motion plan with encoder data only and a motion plan with encoder data and ultrasonic measurements can be viewed by following the links here or in the appendix.

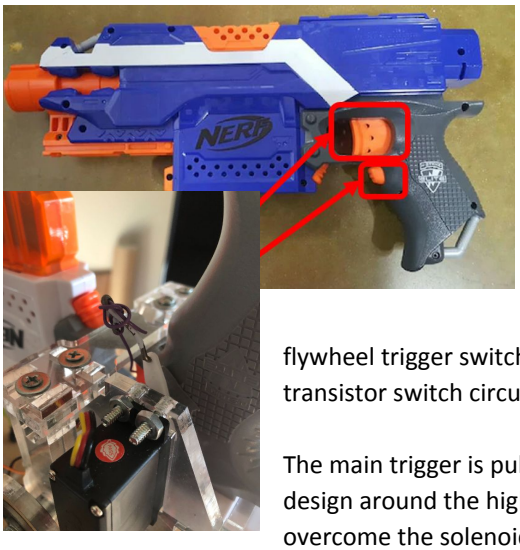


Turret

The turret subsystem consists of the shared Raspberry Pi 3 which operates the turret-mounted Luxonis USB3 Onboard Camera, an Arduino Mega microcontroller to operate the turret motors, two Nema17 stepper motors for pitch and rotation of the turret, a Hitec HS-422 servo motor for operating the Nerf gun firing system, and the rotation and pitch mechanical structure. Overall, the turret responds to commands to rotate, pitch, fire, and self-home. These commands can come from either a master planning program or from the targeting program, which detects targets based on color using the Luxonis camera.

Actuators

The turret was designed to allow controlled movement in pitch at rotation axes. To enable this, we used two Nema17 stepper motors, driven by two Pololu A4988 stepper motor driver boards. The board's step and direction pins are directly connected to an Arduino Mega's digital pins. The boards take 11.1V directly from the robot's Lipo battery, and 5V from the Arduino Mega. The Pitch axis is actuated by having an 8mm acme-threaded rod rotate to propel a lead screw up and down vertically, in a similar method to a 3D-printer's Z-axis. The rotation axis is actuated by a stepper motor direct-driving the smaller of two gears in a 1:4 ratio. The turret platform is bolted to the larger of the two rotation gears. The turret subsystem is described in more detail in Scott Scheraga's graduate project report (ref).

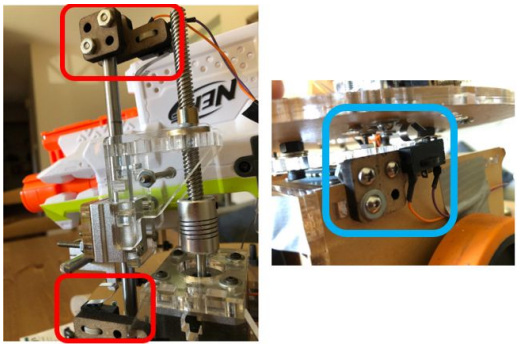


In order to fire the Nerf Gun remotely a number of modifications needed to be made. To remotely spin the Nerf Gun's internal flywheels, the existing four AA batteries in the gun were kept along with the majority of the gun's existing internal electrical circuit. Mechanical locks inside the gun were removed along with a jam-door switch that would occasionally prevent the gun from firing. We connected the TIP120 Transistor-based switch circuit in parallel to the existing flywheel trigger and controlled it with the Arduino Mega. We also repositioned the flywheel trigger switch within the gun housing so that the switch would not be depressed accidentally. The transistor switch circuit layout can be found in the appendix.

The main trigger is pulled by a servo instead of a solenoid because with a solenoid, we would have to design around the high voltage involved, and would need to develop additional mechanical linkages to overcome the solenoids' short "throw" distance. The team utilized a Hitec HS-422 servo motor because we had one already on hand and, from initial tests, the servo proved to have an appropriate amount of torque. The physical trigger-pull mechanism consists of the servo mounted close to the main trigger, a coat hanger-wire extension to the servo horn, and a small diameter "trigger wire" that stretches from the extended servo horn, across the front of the trigger, and around to the other side of the gun. When the servo arm rotates away from the main trigger, the wire is pulled taught, which mechanically pushes a Nerf Dart between the flywheels, launching the dart. In order to fire the gun, the turret Arduino's code includes a 3-second firing sequence, that involves spinning up the flywheels to full speed, rotate the servo away from the trigger, then stopping the flywheels and returning the servo to its original position.

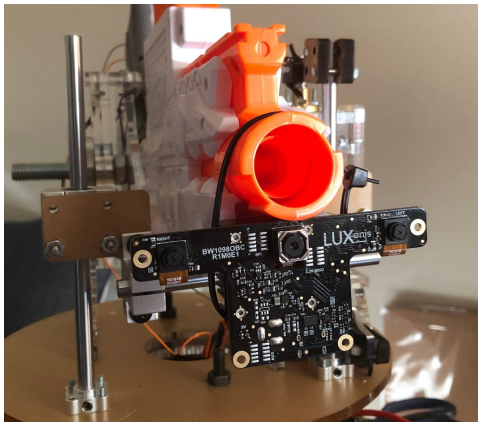
Sensors

The turret subsystem uses microswitches on the turret rotation and pitch mechanism as well as a turret-mounted Luxonis USB3 Onboard Camera. The pitch microswitches highlighted in red in the left image, are used to determine when the pitch mechanism is at its maximum and minimum limits, and software checks switch triggering at every step the stepper motor makes. The rotation mechanism microswitch, shown in the right image, circled in blue is triggered by a low-hanging bolt from the circular turret platform. It is used to "home" the mechanism. Each of the switches are configured as pullup switches.



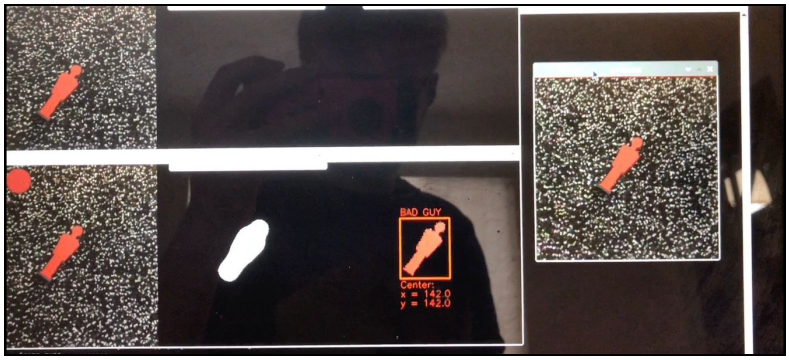
The Luxonis camera is unique in that it has two capabilities a traditional web camera or Pi camera lacks: it can measure distance to objects in view and it can perform complex object recognition algorithms onboard. In a project tasked with identifying targets and firing projectiles at them, these capabilities seem intrinsically valuable. However, the drawbacks of the camera are that it is still very much in a prototype stage, so it is nontrivial to use these capabilities. Other choices for targeting cameras included a Pi camera, a web camera, or a Pixy 2 camera. The web camera would function like a higher resolution Pi camera, and these platforms would simply use color detection built on OpenCV to

identify targets. The Pixy camera has an interesting and very user-friendly color detection software, and significant effort was put into researching and testing the Pixy. However, the Pixy is difficult to extract extra information from (outside of that capability provided by the manufacturer).



Despite the Luxonis' drawbacks, it was chosen in the hope that a training pipeline would be developed by the manufacturer by the time the camera was needed for targeting. This pipeline would theoretically allow use of the onboard resources of the camera for targeting and also give distance information to the target. While not absolutely necessary for this project, the distance information would be valuable if this project were extended to long-range applications. The Luxonis camera also uses the OpenCV library, so, while more difficult to use, if the training pipeline was not available OpenCV could still be used to perform color detection. The training pipeline was developed late in the semester, after color detection software had been implemented on the camera for targeting. Furthermore, the pipeline required additional background knowledge that would have significantly impacted the project schedule, so color detection was used for targeting.

The overall targeting logic is fairly simple. If the camera sees red, then it will label this object as an enemy target. If it sees green, it will label this object as a friendly object. The Police academy course was specially designed such that there are no other red or green colors on the course outside the targets. The Python script developed runs on the master Raspberry Pi computer and uses OpenCV to perform color detection. This software uses the HSV (hue, saturation, value) color parameters to isolate red and green.



The program flow is as follows: first the program takes the image frame information from the camera in the form of a NumPy array and copies this into two new frames. Red and green color detection is then performed simultaneously, with one color performed on each frame. The color is isolated by comparing the HSV values of each pixel to the sets of red and green HSV values. Only the pixels of the original image inside these sets are kept as nonzero values, which are then displayed as the final sensed color. To determine the location of the target, the center, maximum, and minimum pixel locations are found. These values are drawn on the final image displayed to the user. The program also includes a wide variety of debugging and tuning tools, including active color tuning by user-friendly trackbars as well as a host of command line options to configure the targeting script.

Control

The turret must act on the targeting information by moving to point at the desired target. The simple PID controller used requires the user to set a desired point and to tune the proportional, integral, and derivative gains. This will allow the turret to gently home in on the target. This system passes velocity commands to the turret microcontroller which would become smaller and smaller as the turret approached the target.

As the turret moves, the camera, which is attached to the bottom of the Nerf gun launcher, moves with it. So, the turret will fire when the target is roughly in the center of the camera's frame. The actual pixel location fired at can be adjusted with experimental testing. If we assume that the center of frame is the desired firing location, and that the target is initially in the bottom left corner of the camera's view, the control flow will be as follows: the PID will detect the difference between the target's pixel location and the pixel setpoint, and will give a larger value directing the turret to pitch up and to the right. In the next frame, the PID will detect that the target has moved closer to the setpoint and will give a smaller value to direct the turret toward the pixel setpoint. When the

target location has sufficiently overlapped with the setpoint location, the turret is given a command to stop homing in on the target and fire the projectile.

Decision Making

The targeting and control Python script initializes when called by the master script. It starts the camera and checks the initialization variables to determine its PID constants, the color it's detecting, and the initial azimuth and angle to turn to. It then begins performing color detection at between 2 and 5 frames per second. In each frame, if it does not see the desired color, red, it will not send any commands to the turret and will simply timeout and return to the master program after some time. If it does detect the desired color, it will perform the PID calculations to create the velocity commands. Then, it will create a command list to send to the turret which directs it to pitch and rotate based on velocity commands. When the target is sufficiently close to the firing location, the command list will change to direct the turret to stop moving and to fire. The program shuts down after firing and returns to the master program.

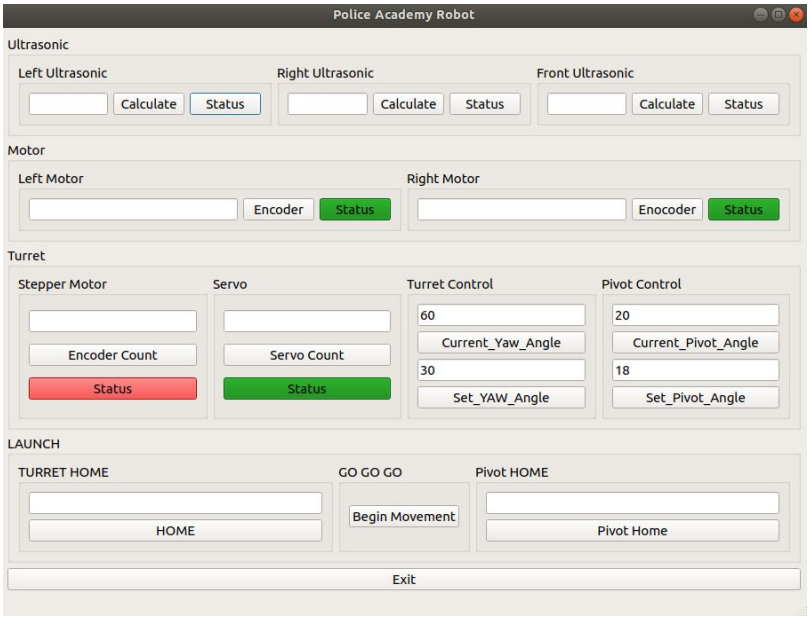
Integration

Communication

The chassis Arduino Uno and the turret Arduino Mega communicate with the Raspberry Pi using the I^2C communication protocol with the Pi as the master and Arduino as the slave. For the chassis, the Pi sends two single byte integer commands between [-3, 3] to accelerate translation or rotation by the specified PWM amount. Additionally, a brake command can be sent to stop both motors. For the turret, the Arduino is sent a 9-element command list of integers which follow a format that allows a wide variety of rotation, pitch, and firing parameters to be changed. The 9 command variables are global variables in the Arduino program, and whenever a new command comes in, the program is interrupted and these global variables are changed. This command list format allows functionality for a variety of auxiliary programs which can command the turret to rotate and pitch by specific increments instead of by velocity commands, as well as to home itself and fire outside the normal targeting sequence.

Graphic User Interface

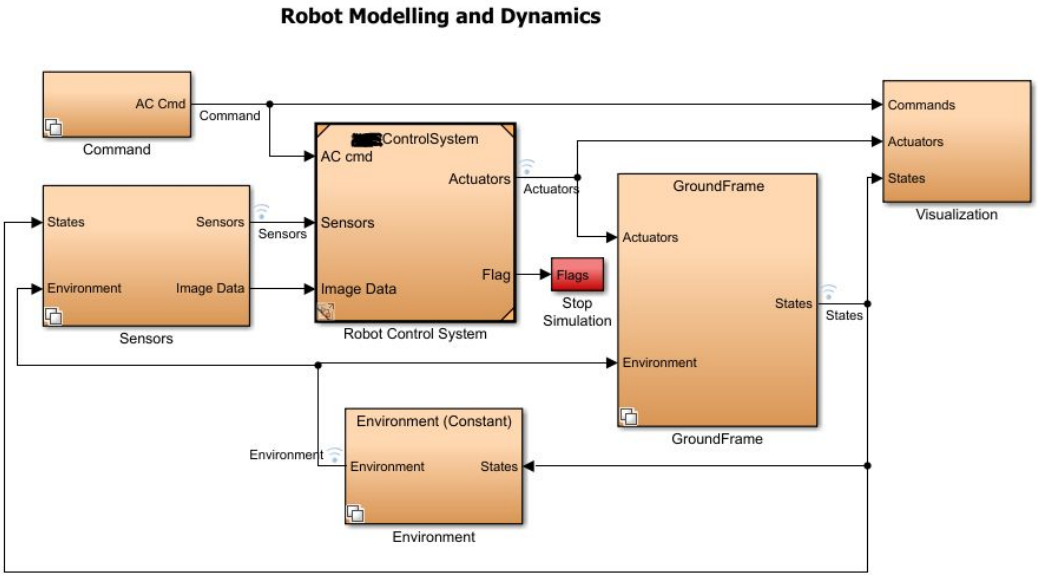
Qt is a set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. PyQt5 is a comprehensive set of Python bindings for Qt v5. Our Graphic User Interface was created to operate the robot during on traversal of the robot for fine tuning and sensor fusion and tracking. Information easily accessible by the GUI at the conclusion of the project includes: current ultrasonic readings and the status of the ultrasonic sensors and current encoder values and encoder status. Commands available allow the robot to begin navigating the map.



Robot Operating System

Robot Operating System (ROS) was planned to be used as a robotics middleware. Although ROS is not an operating system, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. We had initially planned to use ROS to move our robot, but due to ROS latencies and extremely heavy computation requirements we shifted to simpler methods.

Sensor fusion for the ultrasonic sensors to develop pose and odometry tracker are parallel processes which were planned to be undertaken using the Robot Operating System initially. A constant transform broadcaster is generated between the wheels and the base_link. The observed ultrasonic readings are converted with respect to the center of the robot to develop a transform tree. The transform tree generated was planned to be used as the state machine keeping track of the targets detected and these flags are transferred to the base_link. The base_link transform is the pose estimator for the robot. This commands the turret based on the camera target transformations changed frame to the direction and publishes on the ROS_Master topics.



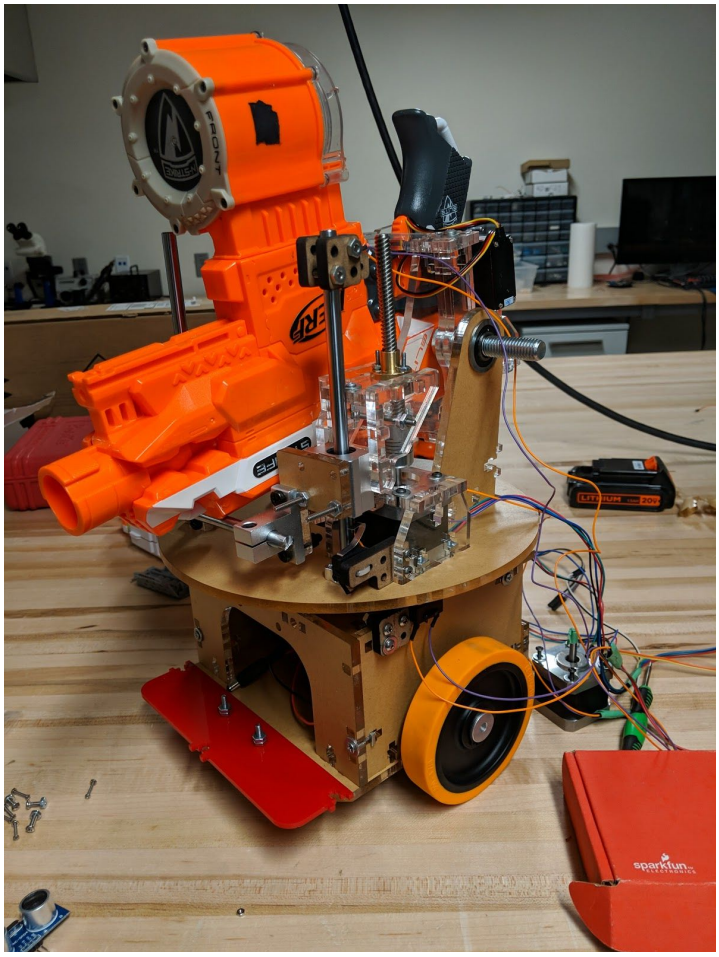
The Odometry is generated on a threaded algorithm which keeps track of the cumulative ultrasonic scan and the encoder data to publish on Odom_Twist Messages. This Odometry is further published onto the landmark generator for Adaptive Monte Carlo Localization. These landmarks generated were planned to be used for correcting the odometry and reset post object detection. Integrating the Joint State Variables of the robot used on the actual map, to see implementation of path planning on the Current Map of the Police Academy Training course. All of this would be recorded by creating a Pipeline to implement and boosting from a Solidworks to Universal Robot Descriptive Format (URDF) Converter.

Results

The AutoDesk Inventor CAD models which were built can be found on GrabCAD in the link in the Appendix. All structural parts were lasercut from acrylic. The robot is able to navigate the map with predetermined commands based on encoder and ultrasonic sensor feedback. However, the robot would need to start from a known position because localization was based on the original map. The robot can accept commands to go forward or backwards a specific number of inches, and to turn left or right a specific number of degrees. Although the robot can accept commands to both turn and move forward at the same time, the path taken would result in the robot running into walls on the course. Therefore, turns and translation were commanded separately. All forms of motion were controlled by different PID coefficients to allow for performance to be tuned independently.

The turret subsystem performed half of its capabilities successfully after integration. The color detection and targeting program was able to successfully identify targets (see video link in Appendix) and pass correct command lists to the turret Arduino over I^2C. However, its capability was limited to a single green and red target at once, and the program automatically assumed that the target would be hit by the projectile after passing the “fire” command to the turret. This functionality should be sufficient for the simple Police Academy course.

The turret control Arduino code was reliably able to interpret commands and was able to perform the “home” and “fire” commands repeatably and successfully when integrated. However, when passed velocity or incremental rotation and pitch commands, the turret performed unexpectedly. Further issues persisted in I^2C communication, with “home” and “fire” command lists being sent without difficulty but with rotate and pitch commands causing the I^2C bus to crash and communication to stop. During the testing phase prior to integration, these commands were received and parsed correctly, so the cause of this crashing is, as yet, unclear.



Future Work

The motion planning could be modified to implement a road map online for the course instead of being developed offline and hard coded. This way, instead of dictating the specific motions to take, the robot will simply read a waypoint, conduct a search of the road map using a search algorithm such as A*, then follow the plan. This would make the system more user friendly, more robust to failure states since the robot could easily return to the starting area, and allow the to autonomously determine the next way point probabilistically based on where it has or has not seen targets. This would require global state tracking instead of the relative state tracking currently implemented, which would further improve the robustness of the system to perturbations between maneuvers.

The targeting program should be modified to include edge detection capability to be able to tackle multiple targets in one frame. This functionality should detect separate targets by determining if they are one contiguous structure or not. It should also have a different method of target strike confirmation. For instance, the program should check to see if it sees the number of blue areas corresponding to the number of targets. These blue areas are visible only when the targets are successfully struck. Furthermore, if nothing is detected by the camera, it would be useful to implement logic such that the camera directs the turret to sweep around and check for targets instead of simply timing out. The Luxonis training pipeline has also been further developed, and may, in the future, be more accessible to students. This software could be used to take advantage of the Luxonis camera’s unique onboard processing and ranging capabilities.

Appendix

Tips for Future Students

This is a short term project with relatively low-cost components. It is better to quickly build something that works okay than to spend a lot of time designing something that is theoretically perfect. There will be unanticipated issues, hardware will fail unexpectedly, and equipment won't work together as expected. Quickly producing a physical system then iterating quickly through updated designs will improve the likelihood that the final product will be able to compete and look good doing it.

It may be tempting to try to design the absolute best project that you think is possible, but it's far better to start simply and add additional layers of complexity. For example, before implementing PID control on our turret, we may have been better served by initially performing simple rotations and pitches, even if they were not accurate enough to hit the target.

It is important to not only plan out wiring, but to also account for ease of access and the space components will require. If you will occasionally need to hit reset on the Arduino, it shouldn't be buried under a bunch of other components. This also means accounting for the orientation required to plug in equipment and how much you can bend cables. If you need to plug in USB cables and a power supply to the Raspberry Pi, there needs to be space for those headers and you need to be able to reach in and unplug them as needed.

Using a perfboard once an electronic circuit is designed will vastly improve the ease of use and help to avoid hardware issues associated with breadboards. Adding LEDs to show when a board is powered helps with troubleshooting and having simple connectors helps with communication. Instead of connecting six loose wires from the motors to the Arduino, plugging the motor into a custom built Arduino motor shield meant there were only two ways for us to physically plug in a motor. By adding markings to both it meant there was one clear one to do it. This saved time in troubleshooting previously solved problems.

For future students working on computer vision, when using hue, saturation, and value parameters, it's useful to note that some colors, like red, consist of two separate HSV areas, while green is only in one area.

Furthermore, if you don't have extensive computer vision experience, it's more useful to build your logic/control around a simpler device which does most of the thinking for you (e.g. the Pixy 2 camera). It may require more creative logic, control, or use of the manufacturer's pre-written software, but it will save valuable development time building a system from the ground up that will require similar creative thinking anyway.

Inter-device communication is very complex and prone to failure. Use serial communication (TX and RX UART communication) wherever possible for your application. I^2C and SPI protocols have significantly less users and thus have less resources available for debugging.

Links

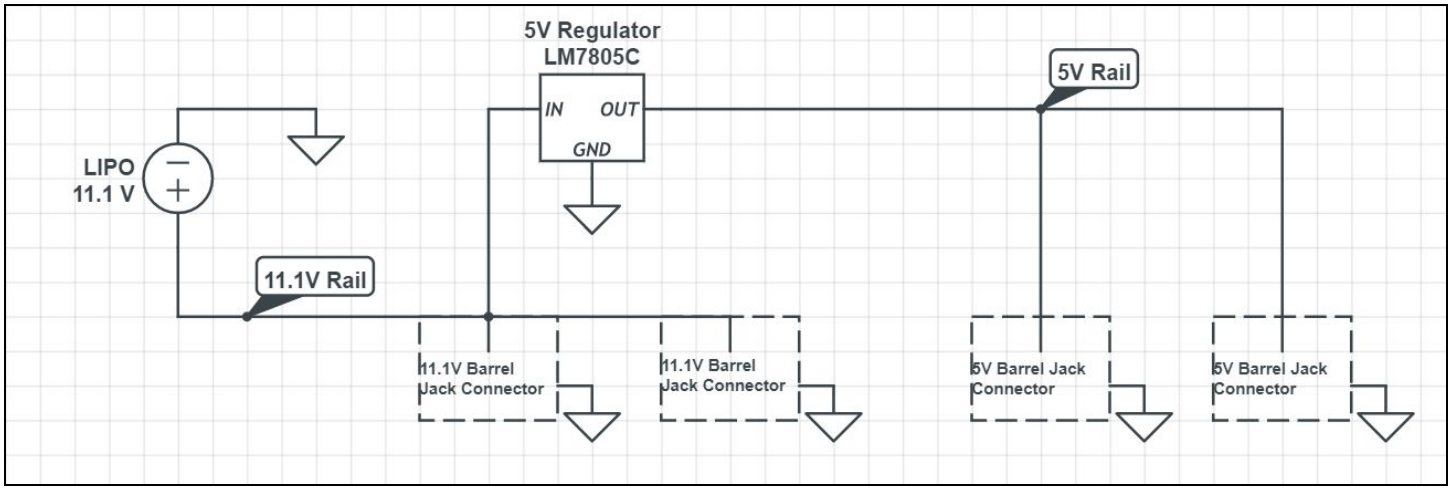
GitHub Repository:	https://github.com/jackcenter/Police_Academy.git
Encoder Navigation Video:	https://youtu.be/trYxAFpn6zo
Encoder and Ultrasonic Navigation Video:	https://youtu.be/AMMQgCAscWY
Red and Green Target Identification:	https://youtu.be/vzmXMASyIJ8
Nerf Flywheel Circuit Layout:	https://forum.arduino.cc/index.php?topic=143263.0
Stepper Motor Driver Circuit:	https://www.pololu.com/product/1182
Mechanical Drawings:	https://grabcad.com/library/mechatronics-final-project-police-academy-robot-1

Bill of Materials:

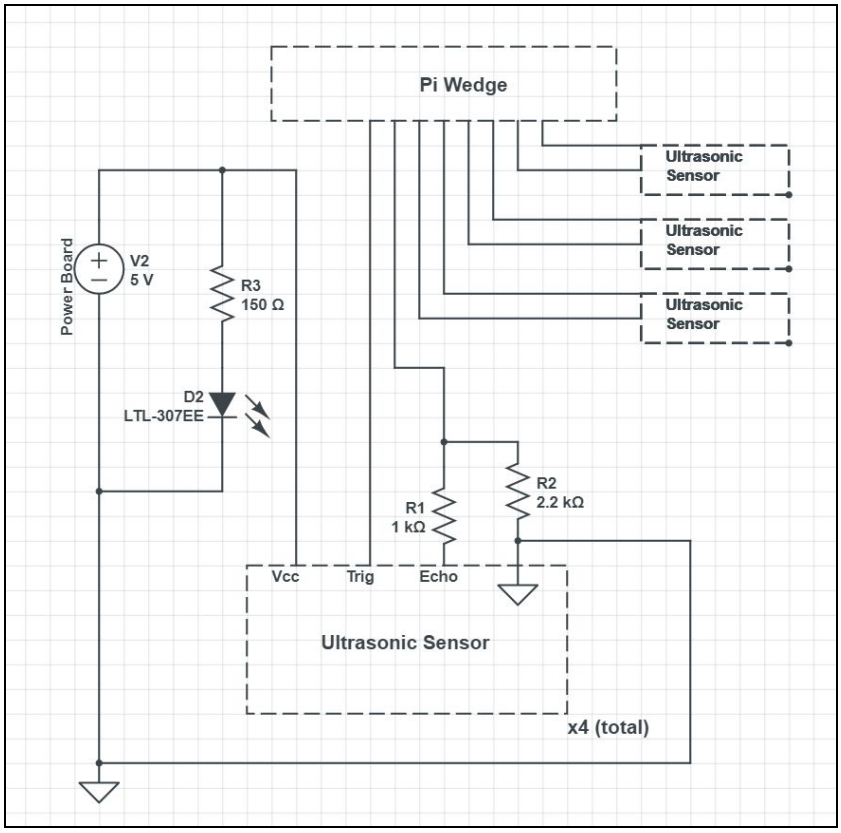
Borrowed from Class/University	**qty**	
11.1V 4Ah Lipo Battery	1	
Arduino Uno	1	
131:1 Metal Gearmotor 37Dx73L mm 12V with 64 CPR Encoders	2	
Ultrasonic Sensor	3	
SparkFun Dual TB6612FNG Motor Driver	1	
Luxonis USB3 Onboard Camera	1	
Many screws and nuts (From robotics lab)	n/a	
USB cable	1	
Borrowed from Personal Equipment	**qty**	
Arduino Mega	1	
Whyachi MS-05 Switch	1	
Nema17 Stepper motor	2	
Microswitch	3	
Pololu A4988 stepper motor driver	2	
5V 2A Cell Phone Battery Charger	1	
Purchased	**qty**	**Total Cost ($)**
Acrylic Sheets	4	46
3D Printer Hardware bundle pack (shafts, threaded rod, lead screw, mounts,anti-backlash shaft adapter etc)	1	33.99
Sunluway ID 1/2" x OD 1-1/8" Flanged Ball Bearings	2	7.99
8mm Bore Set Screw Hub -servocity	2	4.99
5mm Bore Set Screw Hub -servocity	1	4.99
Lazy Susan Bearing- Mcguckin	1	5.5
.5" Bolts- Mcguckin	2	6
2x Banebots T81Hub, 6mm Shaft, and 2X Banebots Wheel 3 7/8In x 0.8" Hub Mount 40A, Orange	1	21.59
Ball caster- Mcguckin	1	12
Nerf Stryfe Modulus	1	32.4
	TOTAL:	175.45

Circuit Diagrams

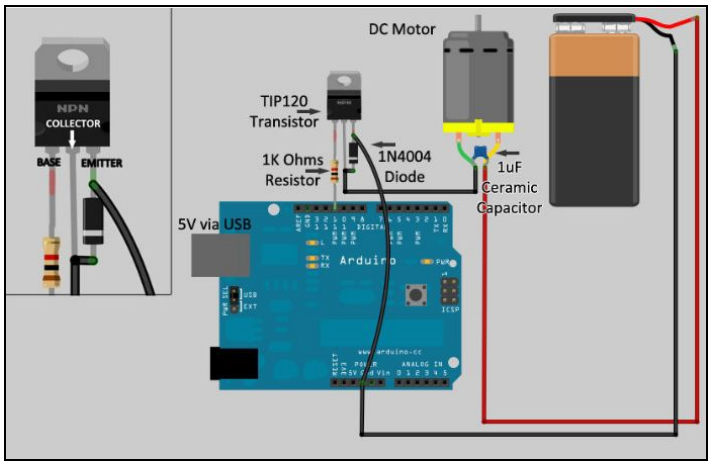
Power Regulation and Distribution Board



Ultrasonic Circuit Board



Nerf Flywheel Trigger Circuit



Stepper Motor Driver Circuit