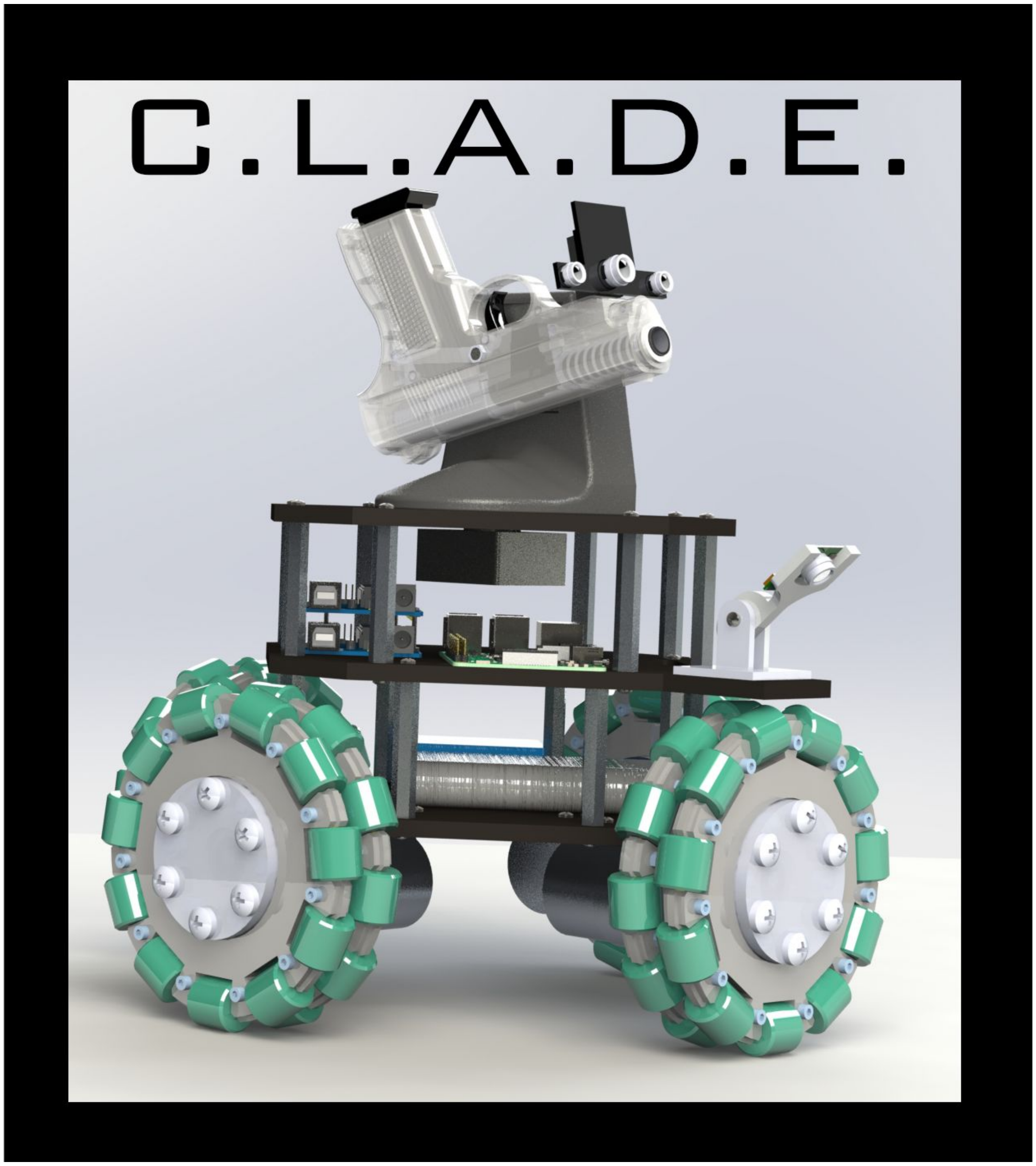
Criminal Locator And Discipline Enforcer

MCEN 5115/4115 Mechatronics and Robotics

Colin Smith, Lea Chandler, Anton Arriaga, Dan Warner, Ellen Rumley



Project Overview

This report describes the design and construction of a robotic system for the Spring 2020 Semester of MCEN 5115/4115: Mechatronics and Robotics at University of Colorado - Boulder.

Mechatronics and Robotics Course Overview

Learn and demonstrate the skills necessary to control mechanical systems with electronics and computers. The course culminates in a team project involving a ‘sensing’ and ‘thinking’ robot that will collaborate with or compete against classmates’ robots. The team project was determined by class consensus.

Project Overview

Apply concepts learned in Mechatronics and Robotics to an open-ended design problem. The task is to design and build an autonomous vehicle that can compete in an adapted version of a police academy shooting course, or ‘Hogan’s Alley.’

Project Schedule

Wednesday, February 19th	Prototypes and design plans due
Wednesday, March 18th	Demonstration of independent subsystem functionality
Monday, April 6th	Demonstration of system integration and functionality
Monday, April 27th	Competition Course End to End Test #1
Wednesday, April 29th	Competition Course End to End Test #2
Thursday, April 30th	Project report submission

Design

Design Process

The team utilized a System's Engineering approach in the design and construction of the robotic system.

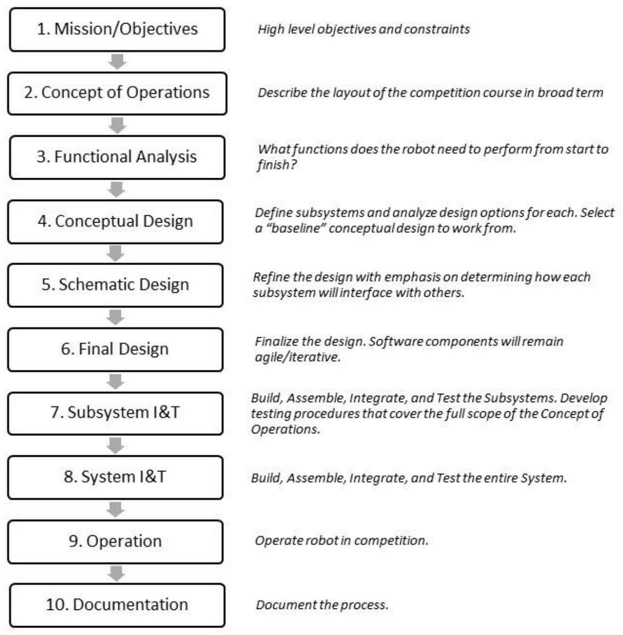

1. Mission/Objectives
High level objectives and constraints
2. Concept of Operations
Describe the layout of the competition course in broad term
3. Functional Analysis
What functions does the robot need to perform from start to finish?
4. Conceptual Design
Define subsystems and analyze design options for each. Select a "baseline" conceptual design to work from.
5. Schematic Design
Refine the design with emphasis on determining how each subsystem will interface with others.
6. Final Design
Finalize the design. Software components will remain agile/iterative.
7. Subsystem I&T
Build, Assemble, Integrate, and Test the Subsystems. Develop testing procedures that cover the full scope of the Concept of Operations.
8. System I&T
Build, Assemble, Integrate, and Test the entire System.
9. Operation
Operate robot in competition.
10. Documentation
Document the process.

Figure 1. Project Systems Engineering Plan

Mission

Objective

Design and build an autonomous vehicle that can compete in an adapted version of a police academy shooting course, or 'Hogan's Alley.'

Project Constraints

A. Budget: $200
B. Must remain within yellow tape boundaries: Dimensions are limited by alley width of 12" and bridge height of 18"
C. Must be internally powered
D. Must be autonomous
E. Six minute time limit per round
F. Maximum of three microprocessors or microcontrollers
G. Maximum of two camera systems

Concept of Operations

The robot shall autonomously navigate the prescribed course in order to find, positively identify, and shoot all 2” x 8” red targets (criminals), while not shooting green targets (good guys). Targets will be located in areas marked by red on the course map. Lanes of travel will be marked by yellow vinyl tape (main thoroughfare width: 16”, smaller alley width: 12”). Intersections will be marked with purple vinyl tape.

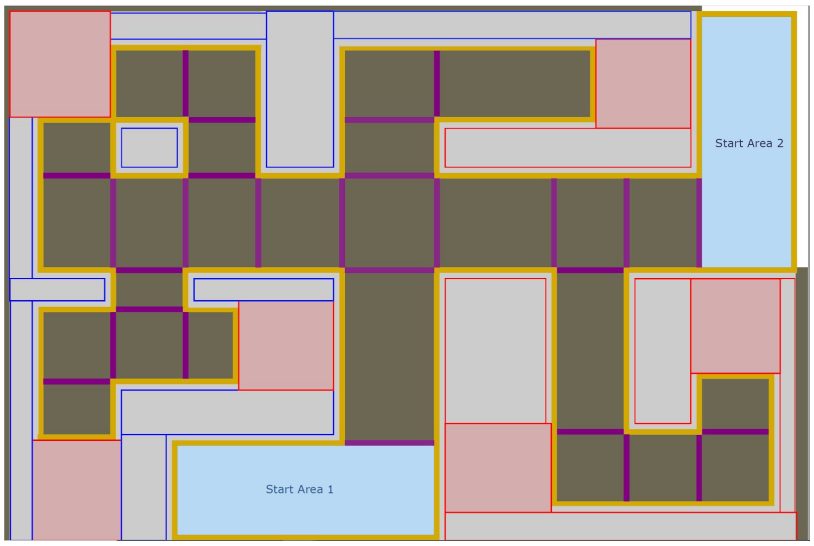


Figure 2. Course Layout

System Functionality

The system shall:

1. Move along two axes (linear: forward/backward, rotate: left/right) required; three axis (add lateral: left/right) preferred.
2. Sense yellow and purple lines with sufficient spatial accuracy to enable turns within project constraints.
3. Sense, identify, and differentiate between red and green targets.
4. Point the projectile mechanism at red targets with sufficient accuracy to hit the target with a projectile.
5. Fire projectiles with sufficient force to knock over targets.
6. Have a projectile mechanism with two axes of movement (rotate: up/down, left/right)
7. Have a durable structure capable of supporting and holding all components.
8. Be internally powered with an operational power-life of no less than 15 minutes
9. Have dimensions not to exceed 11” x 11” x 17” (w x w x h).

Conceptual Design

Analysis of Alternatives

Drivetrain

Trade Space: Omni wheels, standard wheels, two wheels, four wheels, castors
Selection: The team opted to use four omni-directional wheels for two reasons:

1. Availability: Omni wheels were available “Off the Shelf” from previous projects. They could be integrated immediately without external purchase.
2. Flexibility: Four omni-wheels enabled the team to move the robot along 3 axes (X, Y, Z-rot). While Y-axis movement (lateral translation) was not strictly required, it was useful to have as an option when designing the guidance software. If it was not used, there would be no detriment to the robot’s performance. Ultimately, the final guidance software utilized lateral movement to center the robot in tight spaces, as opposed to strictly relying on a combination of x and z-rot commands.

Guidance “Sensing”

Trade space: IR sensors, simple computer vision, complex computer vision (Luxonis depth finding), sonic sensors, Lidar.
Selection: The team opted to use simple computer vision consisting of a Pi Camera, and OpenCV based Python Software for the Guidance Sensing System.

1. Lidar was ruled out for being cost prohibitive.
2. The team opted to use the Luxonis Camera for the targeting system.
3. IR sensors were ruled out because they would contribute least to the learning process. Modern robots with advanced problems rely on more advanced technology, and the team wanted to work in a more relevant field.
4. Sonic sensors were ruled out before the course was revised to include a significant amount of walls. In retrospect, including sonic sensors would have added significant capability with relatively little effort; however, a robot that relied entirely on sonic sensors was judged to have the same shortfalls as IR sensors - limited learning opportunity.
5. Computer Vision was selected for enhanced capability and learning opportunity. The team was determined to use this project as an opportunity to learn computer vision instead of relying on older technology with limited application to modern tech (self driving cars, spacecraft, etc). The team also concluded that computer vision “should” have greater awareness of surroundings and enhanced capability.

Target “Sensing”

Trade space: Luxonis stereo camera with neural network target recognition and depth finding, single camera (PiCam or USB WebCam) with computer vision using shape recognition and color thresholding.
Selection: The team opted to use the Luxonis camera.

- The Luxonis camera had the potential to offload target image processing from the Raspberry Pi to its own onboard computer using a trained neural network.
- The Luxonis also included a color camera, allowing for traditional computer vision techniques developed in Lab 6 to be used as a backup.
- A disadvantage of the Luxonis is that it outputs image data to a buffer at a constant FPS and will crash if the buffer is filled. This is mitigated by setting a low FPS for the camera and by reading images from the buffer constantly, even if they are not processed.
- Ultimately, development time of the targeting neural network proved prohibitive and color thresholding was used on image input from the Luxonis color camera.

Projectile Mechanism “Shooter”

Trade Space: Airsoft, ping pong gun, nerf projectiles
Selection: The team opted to use a modified airsoft pistol.

- Advantage of airsoft projectiles
    - Compact projectiles reduce jamming.
    - Compact projectiles less affected by drag - more accurate.
    - Greater quantity of ammunition fits within gun cartridges – more forgiving of misses during autonomous missions.
    - Spent projectiles are less likely to inhibit computer-vision-based navigation
    - More easily integrated with minimal mechanical work. Most nerf guns require manual manipulation of a slide to reload each projectile.
- Disadvantages of airsoft projectiles
    - Small projectiles are more likely to miss target
    - Lightweight projectiles carry less momentum, potentially reducing the likelihood of tipping the target. This was tested prior to selection.

Projectile Mechanism “Turret”

Trade Space: <u>Configurations</u>: 1-DOF, 2-DOF; <u>Actuators</u>: Steppers, DC motors, servos
Selection: The team opted for using 2 stepper motors to control 2-DOF (elevation and rotation) of the gun and turret.

- Advantage of stepper motors vs alternatives
    - Precise open-loop position control results in high firing accuracy without a complicated closed-loop controller.
    - High holding torque for stabilizing the gun during firing.
- Disadvantages of stepper motors
    - High power usage increases demands on the battery.
    - Large package size makes it harder to integrate into the turret design.
- Advantages of 2-DOF turret over 1-DOF turret
    - The rotation axis decouples target acquisition from navigation, simplifying the high level navigation logic.
    - The elevation axis broadens the effective range of the robot allowing for both closer and longer target ranges.

- Disadvantages of 2-DOF turret
    - Extra complexity in both code and hardware design
    - More failure points

Conceptual Design

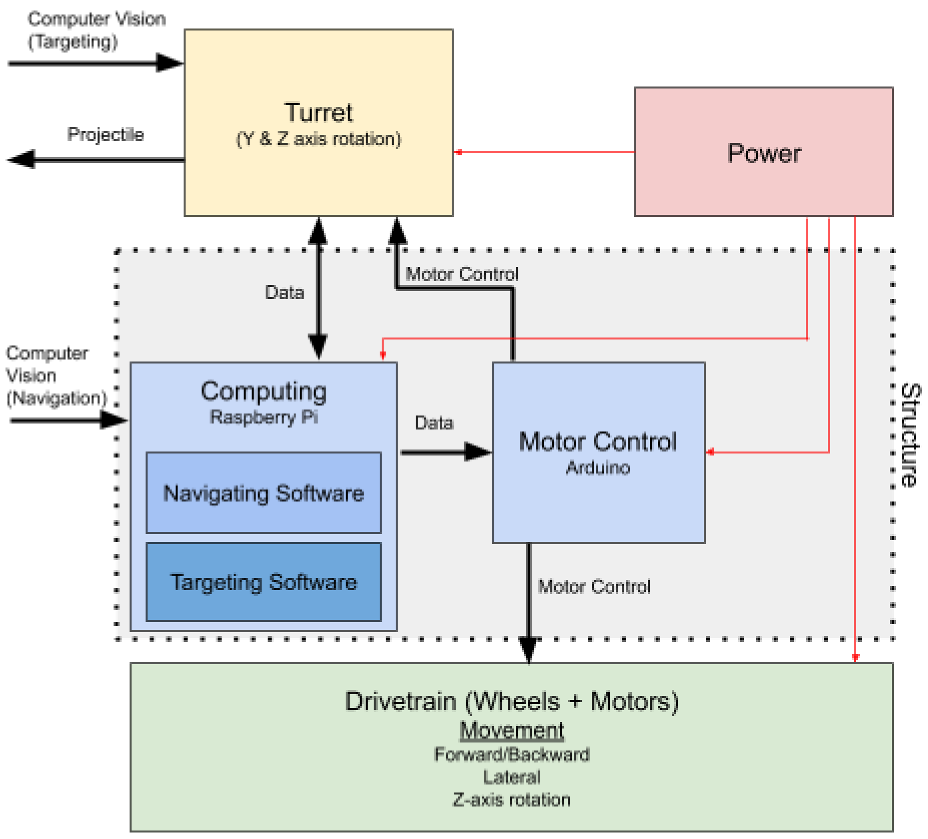


Figure 3. Conceptual Design Plan

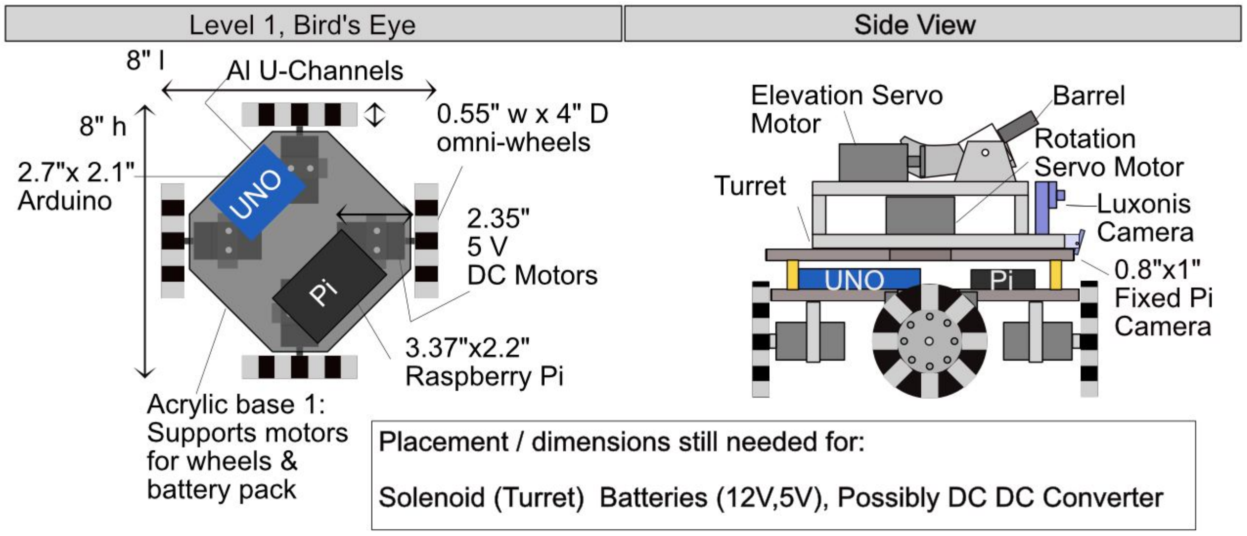


FIgure 4. Detailed Conceptual Design Plan

Final Design

Overview

Structure: Three-Tiered Medium Density Fiberboard (MDF) structure with hexagonal standoffs

Power: 7.4V 2-cell Li-Po Battery: Motors, stepper motors, and Luxonis Camera

5V Rechargeable USB Compact power source: Raspberry Pi, Pi-Camera and Arduinos

Drivetrain: 4 x Omni-Directional wheels

Turret: 2 x Stepper Motors in a custom 3D Printed Housing

Projectile Mechanism: Commercial off the Shelf (COTS) electric airsoft pistol

Operating Software: Python 3

Guidance Sensing: Pi-Camera w/ OpenCV (Python 3)

Target Sensing: Luxonis Camera w/ OpenCV (Python 3)

Primary Computer: Raspberry Pi 3

Motor Microcontroller: Arduino Uno (C++)

Turret Microcontroller: Arduino Micro (C++)

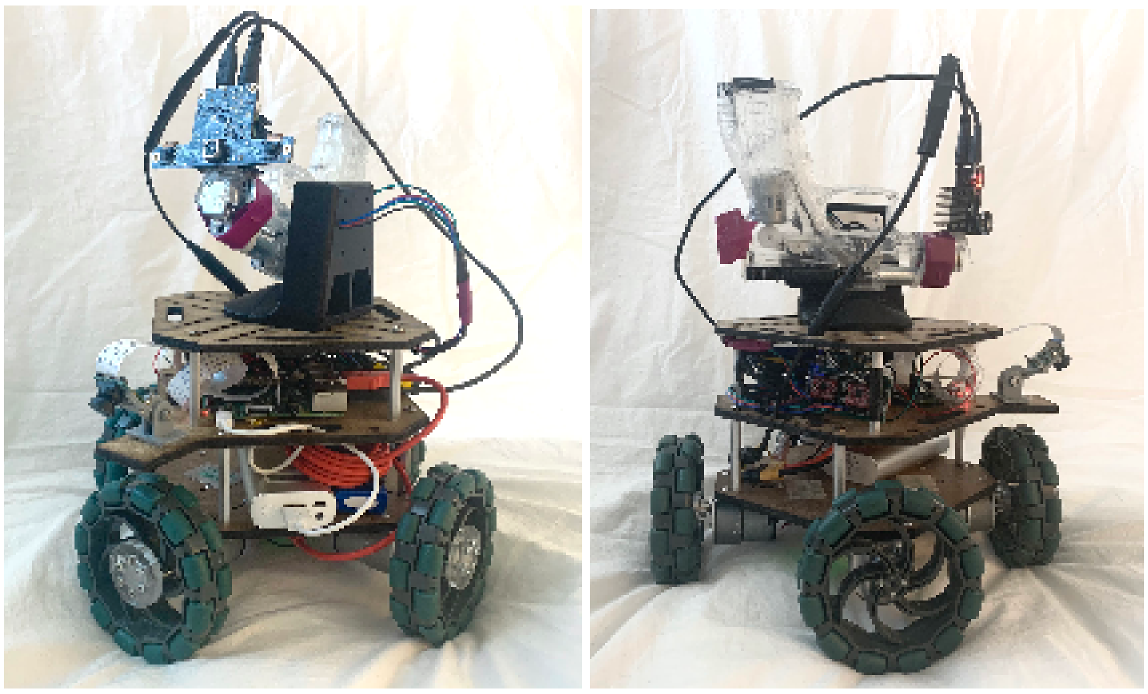


Figure 5. Completed CLADE Robot

CAD Rendered Final Assembly

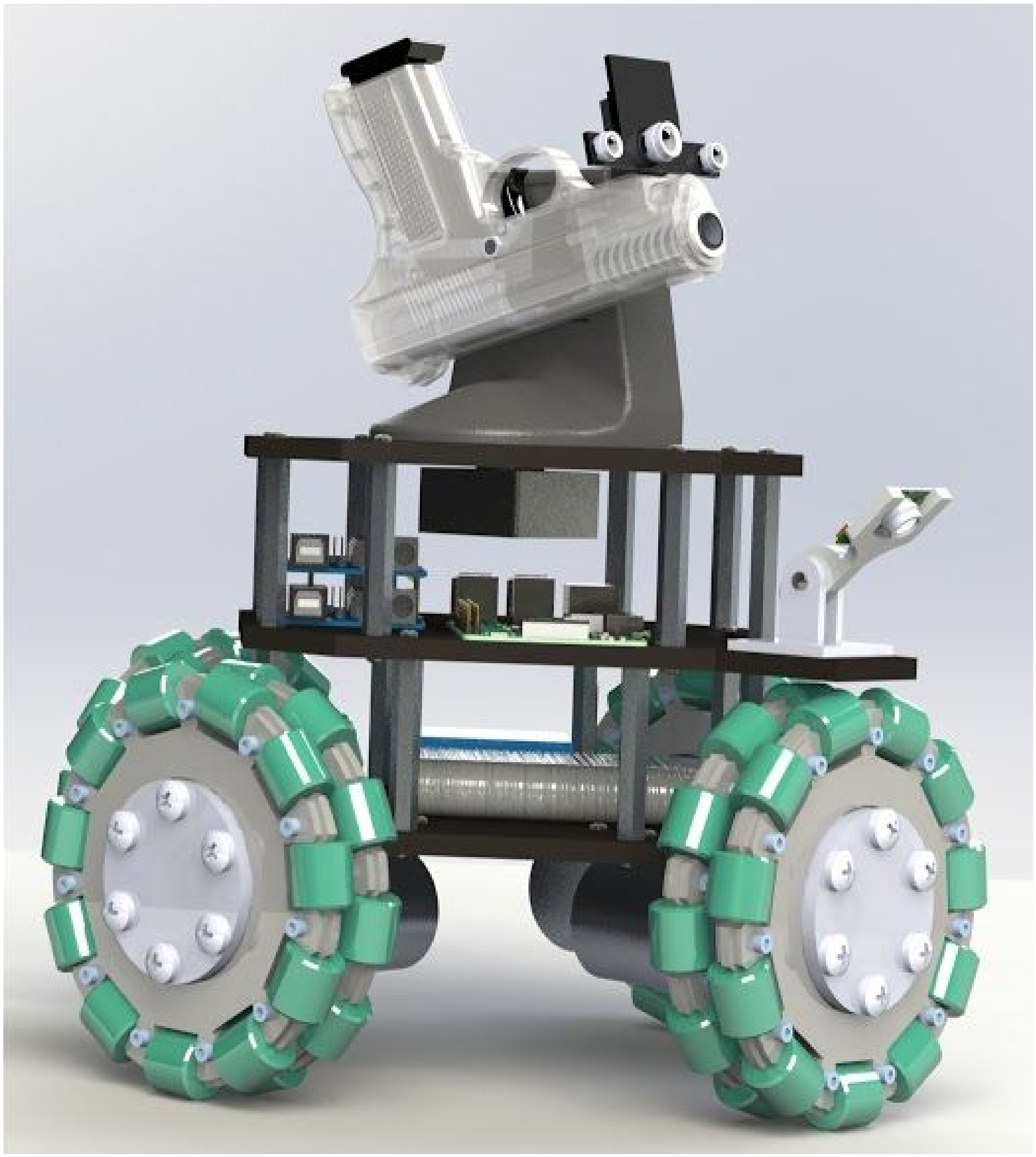


Figure 6. CAD design of robot body rendered on Solidworks - wiring not included.

Assembled Schematic

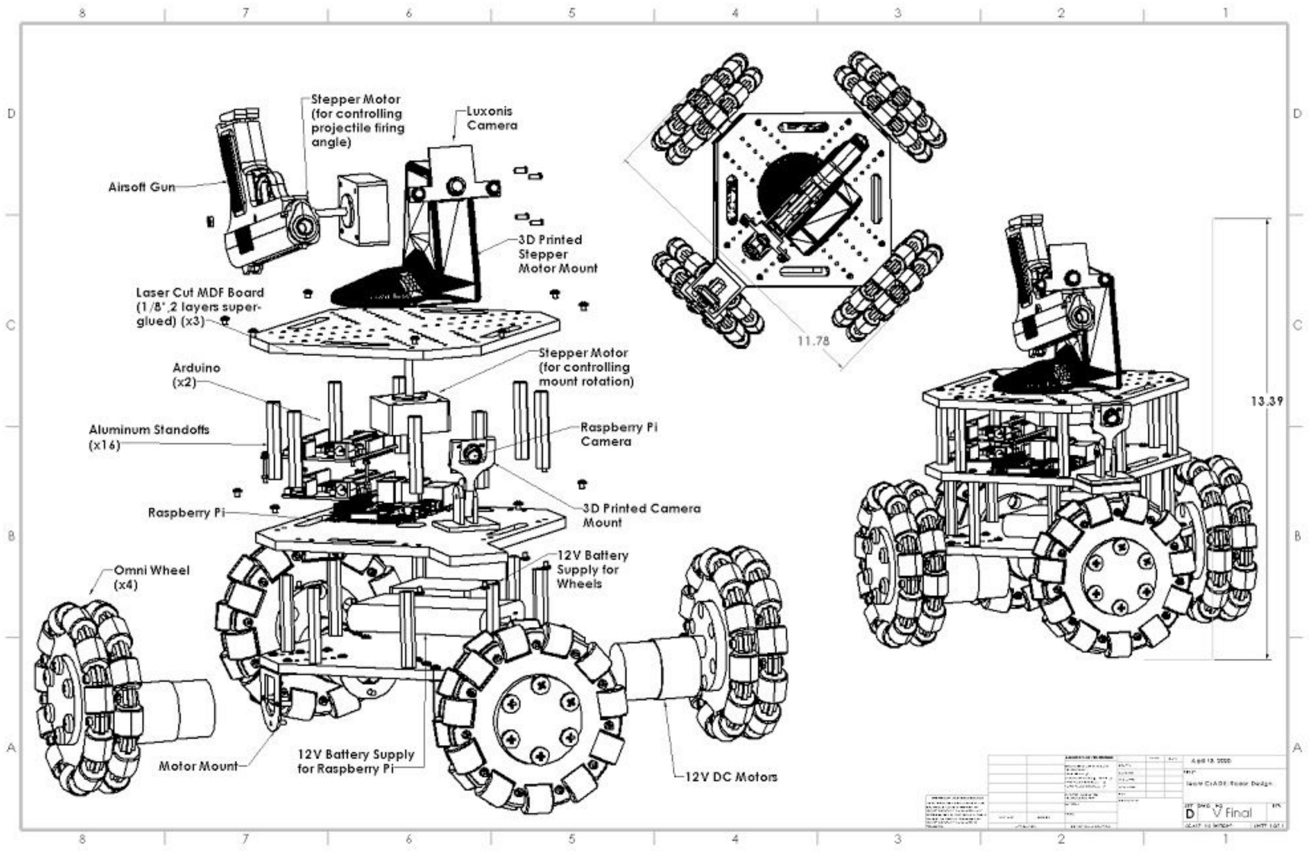

Stepper Motor (for controlling projectile firing angle)
Luxonis Camera
Airsoft Gun
3D Printed Stepper Motor Mount
Laser Cut MDF Board (1/8",2 layers super-glued) (x3)
Stepper Motor (for controlling mount rotation)
Arduino (x2)
Aluminum Standoffs (x16)
Raspberry Pi Camera
Raspberry Pi
3D Printed Camera Mount
12V Battery Supply for Wheels
Omni Wheel (x4)
Motor Mount
12V Battery Supply for Raspberry Pi
12V DC Motors
11.78
13.39
April 13, 2020
Team CLADE Robot Design
D
V Final

Figure 7. Left - Exploded view of CLADE with part labels. Center and right - Top and side views of CLADE with dimensions, respectively. Generated via Solidworks Drawing. (Wires excluded.)

Subsystem Design

Structure

Medium density fiber (MDF) board was chosen as the material for the robot frame due to low cost (~$6 for 3'x2') and ease of modification. Two layers of ⅛" thick MDF were super glued, the glue serving as additional mechanical reinforcement. ⅛" thick MDF was easily laser cutted to desired lengths, and was easily drilled through for quick mounting. The geometry for the main frame was designed using Solidworks.

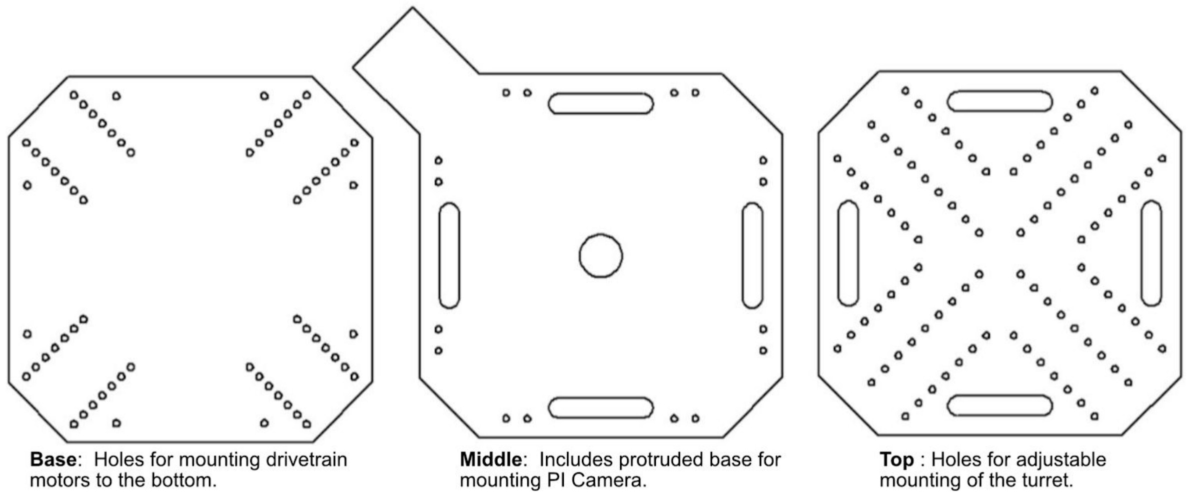

Base: Holes for mounting drivetrain motors to the bottom.
Middle: Includes protruded base for mounting PI Camera.
Top : Holes for adjustable mounting of the turret.

Figure 8. Base, middle and top layers of CLADE body. Constructed from laser cut MDF board.

Drivetrain

The drivetrain consists of four omni wheels mounted on DC motors. The use of four independently controlled omni wheels allows the robot to maneuver forward/backward, left/right, and rotate left/right along the z-axis. The DC motors are controlled with input from an Arduino input through two Arduino motor drivers**.**

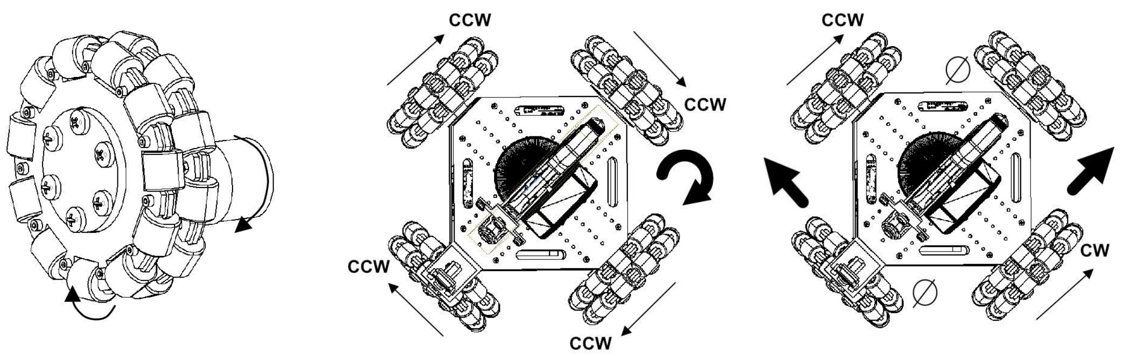


Figure 9. Left: Each wheel has 2DOF. Right: Robot able to maneuver along 3 axes.

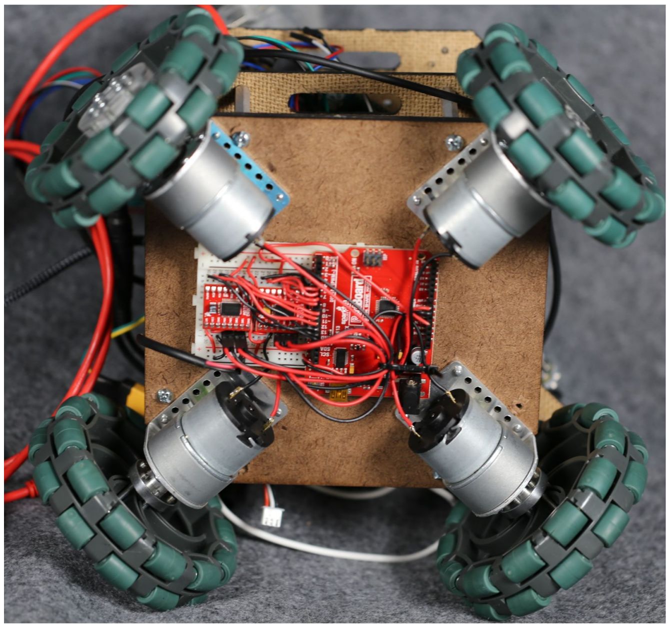


Figure 10. Motor drivers were installed on the bottom of CLADE, where they were immediately accessible to the drivetrain motors, and helped counter the top-heaviness of the turret and gun.

Power

The power subsystem consisted of two batteries.
A 5V USB Rechargeable power supply was used to power the Raspberry Pi. It was stable, long lasting, and compact. The team selected a 7.4V Li-Po Battery to power the remaining components.

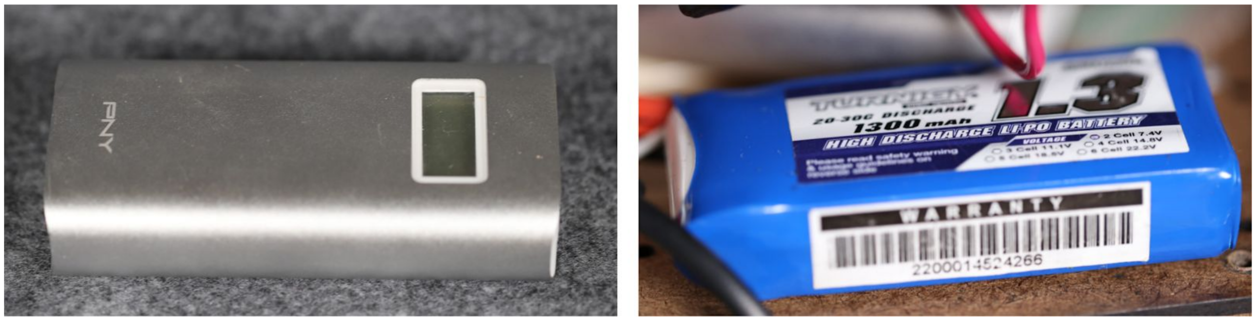


Figure 11. Left - A USB rechargeable battery that supplied power to the Raspberry Pi, which in turns provides power to the Arduino microcontrollers. Right - A 1300 mAh 7.4 V high discharge lithium-polymer battery used to power the drivetrain.

Projectile Mechanisms “Turret

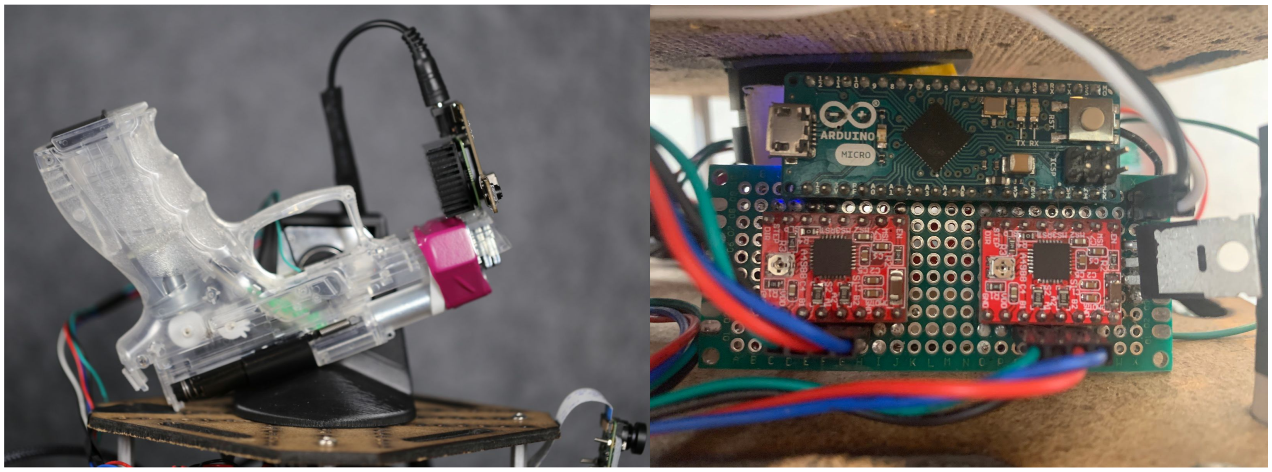


Figure 12. Left: Side view of the airsoft gun used to shoot projectiles at criminals. Attached to the top is the Luxonis camera. Right: Arduino Micro microcontroller and circuit board.

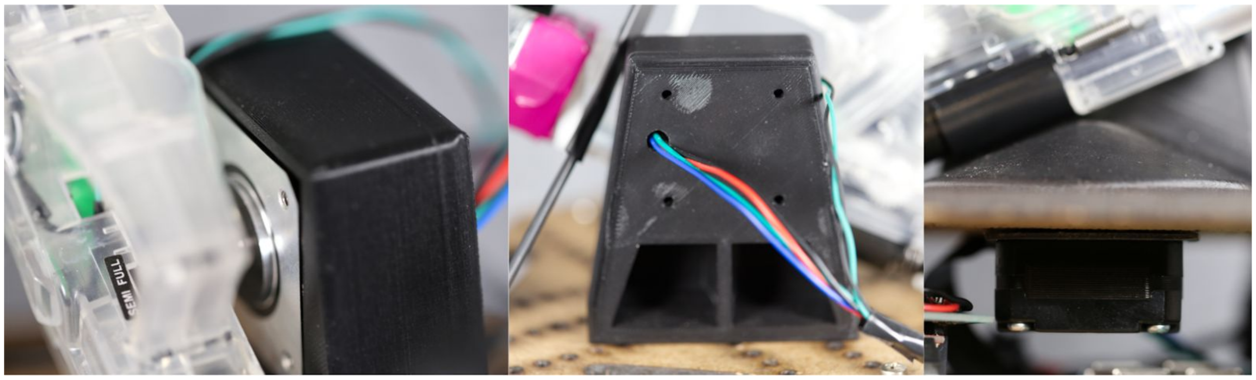


Figure 13. Left - A stepper motor is connected to the base of the airsoft gun to control its vertical motion while tracking targets. Center - A 3D printed base that holds the stepper motor vertically for attaching to the airsoft gun. This base duals as a turret which is swiveled horizontally for targeting using the (right - side view) second stepper motor mounted below the top MDF board.

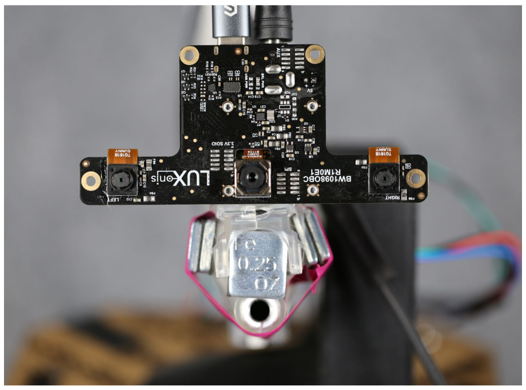
Computing Hardware

Processors

The robot has three microcontrollers onboard: a Raspberry Pi, an Arduino Uno, and an Arduino Micro. The Pi runs the targeting and guidance code, and Uno runs the drivetrain driver code, and the Micro runs the turret driver code.

Cameras

There are two onboard cameras to provide the input for the vision-based targeting and navigation functionality. The first is a Luxonis stereo camera mounted to the barrel of the airsoft gun for targeting, shown in Figure 9. The center of the camera was mounted such that it aligned closely with the center of the gun. Any error in mounting was mitigated by calibration in the software. It requires 5V of power and connects to the Li-Po battery. It is connected to the Raspberry Pi via USB to send and receive data.



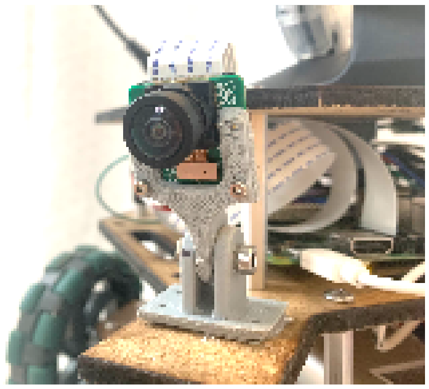


Figure 14. Left: The Luxonis camera mounted to the gun for targeting. Right: Guidance Pi-Camera with wide angle lens

The second camera is a PiCam with a fisheye lens that is mounted to the front of the structure at a ledge over the wheels. It is angled towards the ground to view the path in front of the robot. The PiCam is connected to the Raspberry Pi via a ribbon cable to send data and receive power.

Software

The software for the robot includes Python code for the Raspberry Pi 3B+ and Arduino code for the Arduinos. The Python code consists of three core modules: the driver module, the targeting module, and the guidance module. The Arduino code consists of two modules: the turret firmware and the drivetrain firmware. The source code for each module can be found in the supporting documentation

Software Driver

The main software driver is responsible for initializing the serial ports for communicating with the Ardinos and passing those write descriptors to the commanding modules. It then alternates calling the guidance and targeting modules to process a frame from their respective cameras and issue commands to the Arduinos. The modules are designed so that they retain program control during and maximum FPS processing during important operations: turning in the guidance module and aiming in the targeting module. The flowchart for the driver is shown below in Figure 10.

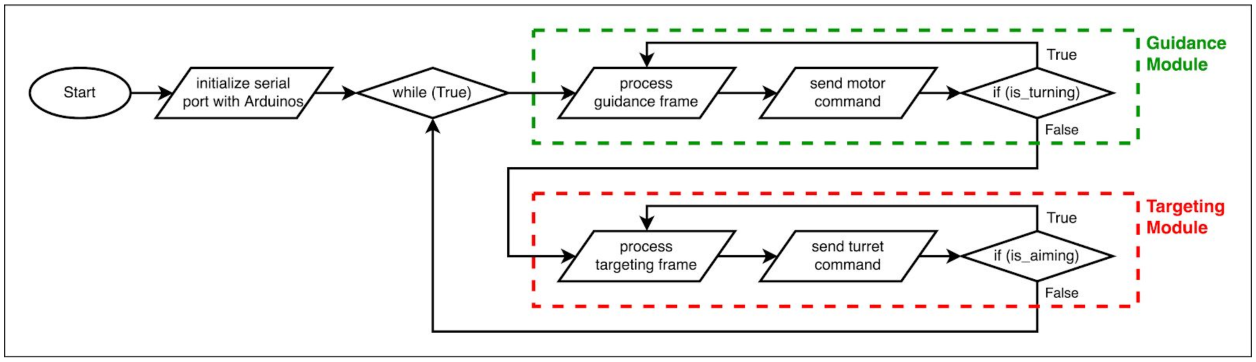


Figure 15. Software driver flowchart

Targeting

Overview

The targeting module uses the Python OpenCV library to process image input from the Luxonis camera. The software will take a single frame from the camera and use image processing to determine if there is a target in the frame. If a target is found, the module enters a loop of aiming and shooting. Otherwise, a command is sent to the turret to return to its home position and the module returns.

Logic

When called, the module initializes a variable IS_AIMING to true and enters a loop that processes images while IS_AIMING is true. To account for blurred frames as the turret is moving to aim, a time variable TIME_TARGET_LAST_SEEN is used to force a found target to "persist" through future frames. In addition, HORIZONTAL_OFFSET and VERTICAL_OFFSET values are used when aiming to account for imperfect mounting of the camera. Finally, a time variable TIME_LAST_FIRED is used to enforce a 3 second limit between firing rounds. A basic flowchart for the targeting logic is also shown below in Figure 11.

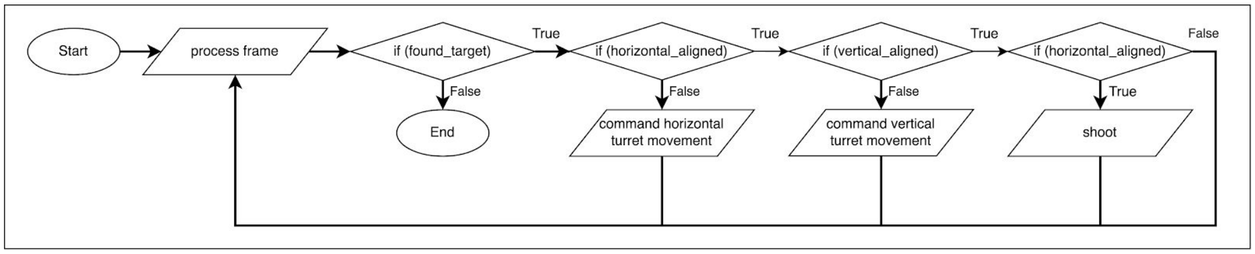


Figure 16. Targeting module flowchart

Image Processing

1. The image is first converted to the HSV color scale. An HSV red mask is defined and applied to the image.
2. A bitwise OR operation is performed to convert the image to grey based on masking.
3. A Gaussian Blur is applied to the grey image to smooth the edges of the boundaries.
4. A black and white value threshold is then applied to the smoothed image to create hard contours.
5. If the size of the largest contour is above a minimum threshold size, the program decides that a target has been detected and TIME_TARGET_LAST_SEEN is set.
6. The horizontal and vertical distances, in pixels, between the center of the target and the center of the frame is then calculated.
7. If a target is not detected and TIME_TARGET_LAST_SEEN is less than 1.5 seconds ago, IS_AIMING is kept as true.
8. Otherwise, IS_AIMING is set to false.

Aiming

1. If the horizontal distance between the center of the target and the center of the frame plus HORIZONTAL_OFFSET is greater than some tolerance, a command is issued to the turret to move horizontally.
2. Otherwise, a variable SHOOT_READY is set to true.
3. If the vertical distance between the center of the target and the center of the frame plus VERTIAL_OFFSET is greater than some tolerance, a command is issued to the turret to move vertically.
4. Otherwise, if SHOOT_READY is set to true and TIME_LAST_FIRED is more than 3 seconds ago, a command is issued to the turret to fire and TIME_LAST_FIRED is reset.

Turret Driver

Overview

The turret driver runs on an Arduino Micro and accepts commands over a USB serial connection from the targeting module to command the turret elevation and rotation motors and trigger firing of the gun.

Receiving Commands

Serial communication is initialized with a baud rate of 9600. Valid commands from the targeting module are formatted as three letters surrounded by angle brackets, e.g. “<HOM>”. The commands correspond to the actions: go home, move up, move down, move left, move right, fire, and stop. Characters are read one at a time from the serial buffer and stored in a local buffer. When the terminating marker is received, the string in the buffer is terminated with a null character. If the serial buffer is empty and the last command received is fresh, the last received command is continued until it times out and comes to a stop.

Turret Commanding

A stepper motor object is created for each stepper motor to facilitate commanding using two arduino digital output pins and an RPM of 20 RPM.

If the command string corresponds to a “move left” or “move right” command, the rotational turret is commanded to move 1 step in the correct direction. If the command string corresponds to a “move up” or “move down” command, the elevation turret is commanded to move 1 step in the correct direction. As these steps are commanded, the position of the stepper is tracked by a counter. When a “home” command is received, both steppers are commanded based on these counters to return to their initial locations. Finally, if a “fire” command is received, the code writes a high value to the fire pin for 200ms of time to trigger the gun. The “stop” command is only used internally by the turret driver and commands the turrets to step 0 steps.

Guidance

Overview

The guidance software uses the Python OpenCV library to process images from a wide-angle-lens Pi-Camera. It takes a single frame, filters the image to isolate purple and yellow (separately), plots lines to match yellow (lanes) and purple (intersections) using a Hough Line transform, and issues guidance commands based on rules-based logic.

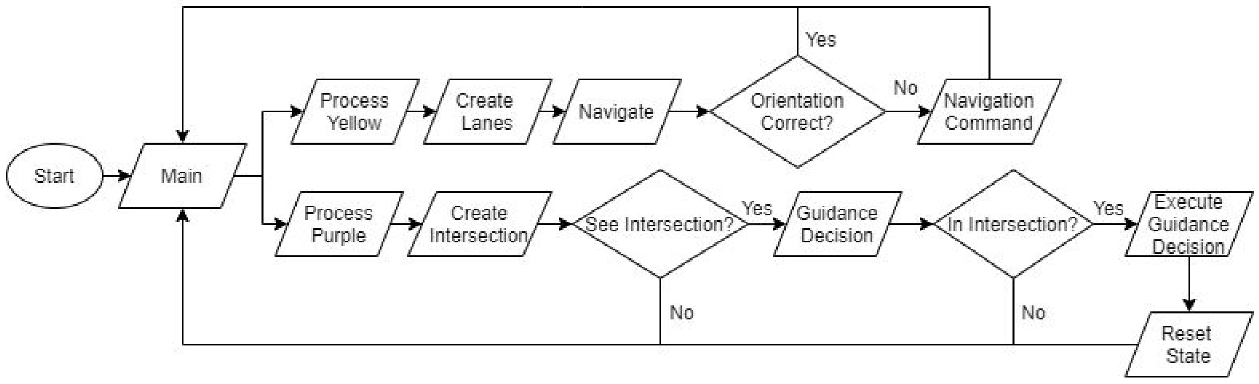


Figure 17. Guidance Module Flowchart

Main Function

The main function loops through sub-functions and controls the “state” of the guidance software. The processing and handling of lanes (yellow) and intersections (purple) is largely handled separately. The guidance software proceeds through the following states:

1. Navigating: Drive within lanes and search for intersections.
2. Guidance Decision: After locating an intersection, determine type and select direction of travel.
3. Intersection: Navigate into intersection.
4. Execute Guidance Decision and Reset.

Image Processing

1. Mask a “Region of Interest”
   - Crop portions of the frame to limit the camera’s field of view
2. Convert frame to HSV (Hue Saturation Value) Colorspace
   - HSV is a simpler colorspace to work in than RGB. The entire color spectrum can be described by a Hue (the color), Saturation (Intensity of color), and Value (Brightness). So, for example, yellow HUE lies between 40 and 70 (out of 360). Variations of yellow lie in the entire extreme of Saturation and Brightness Values.
3. Filter the image to isolate yellow or purple colors
4. Convert frame to grey-scale color space
5. Use Canny Edge Detection to trace an outline around lines
   - This produces an almost entirely black image with white edges marking the outlines of our lanes or intersections. This image is output to the next function.

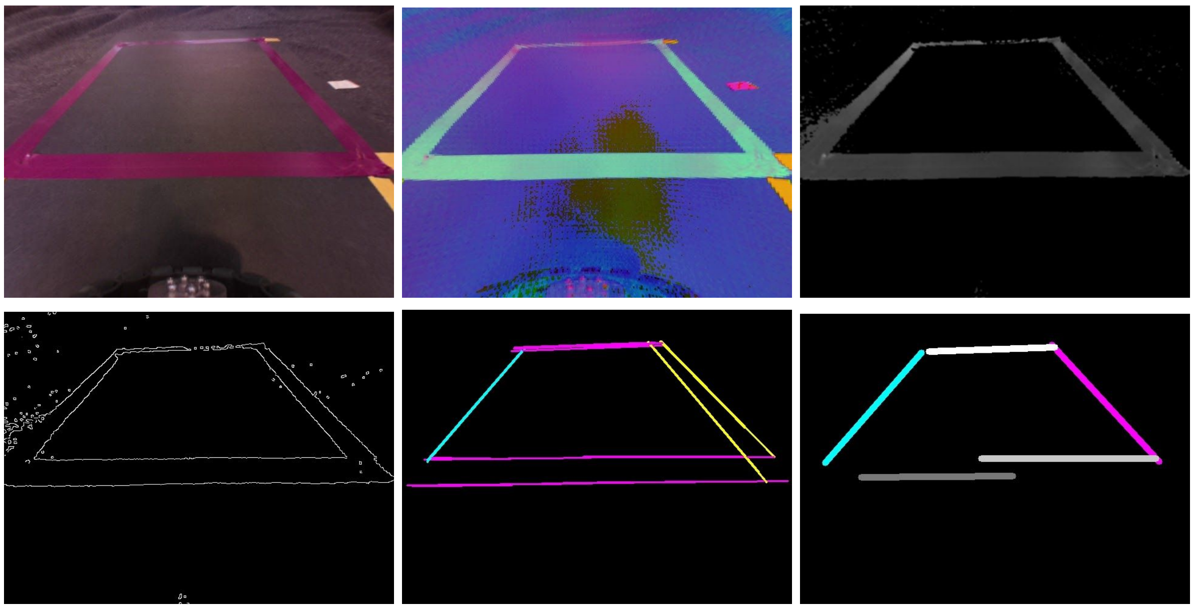


Figure 18.

L to R, Top to Bottom : 1. Distortion corrected image | 2. HSV | 3. Isolate purple/convert to grey
4. Canny Edge Detection | 5. Probabilistic Hough Line Transform | 6. Averaged Hough Lines

Create Lanes / Create Intersections

The Lane and Intersection functions take fully processed edge images that are still represented as pixels and use a Probabilistic Hough Line transform (cv2.HoughLinesP) to transform them into lines defined by start and end coordinates (x1, y1, x2, y2). It proceeds as follows:

1. Hough Line Transform: For an 'edge frame' draw every possible set of lines to describe pixelated lines mathematically, and threshold them based on a set of parameters (length, angle, etc).
2. Categorize all of the hough lines based on slope or region of frame (i.e. left lane, right lane, center).
3. Average the hough lines in each category to produce a single mathematical representation of each lane or intersection line.

Navigation

Using the categorized and mathematically defined lanes, the Navigation function uses a series of rules to define its orientation relative to the lanes. If the robot is not centered or oriented directly down the road, it issues a command to turn or move laterally. When it is centered between the yellow lines it moves forward.

Guidance Decision

When the system "sees" an intersection within a prescribed region of interest, it makes a guidance decision based on the types of intersection lines visible in the frame. The logic prioritizes right turns, followed by straight, and lastly left turns. For example, if the intersection consists of a left, right, top, and bottom line, the robot will decide to make a right turn, and lock that decision into memory until it has proceeded to the center of the intersection.

Execute Guidance Decision

After a guidance decision has been made, the robot continues forward into the intersection. It "counts" the bottom of the intersection as it passes over, and drives forward (by counting seven issued FWD commands) into the intersection. It then commands the guidance decision (LLL, RRR, FWD) for a set amount of time to complete a 90 degree turn or proceed straight.

Drivetrain Driver

Overview

The drivetrain driver runs on an Arduino Uno and accepts commands over USB serial from the guidance module to command the four wheel motors.

Receiving Commands

Serial communication is initialized with a baud rate of 9600. Valid commands from the guidance module are formatted as three letters surrounded by angle brackets, e.g. "<STP>". The commands correspond to the actions: forward, backwards, lateral right, lateral left, rotate right, rotate left, and stop. Characters are read one at a time from the serial buffer and stored in a local buffer. When the terminating marker is received, the string in the buffer is terminated with a null character. If the serial buffer is empty and the last command received is fresh, the last received command is repeated until it times out. Then, all drivetrain movement is stopped.

Drivetrain Commanding

Each motor is controlled with 2 digital logic pins and 1 PWM pin on the Arduino Uno. The digital logic determines the direction of the motors and the PWM maps to the motor power. LOW/LOW on the digital logic pins puts the motor in an unpowered state. LOW/HIGH will command the motor in a direction. HIGH/HIGH will hold the motor in place with electrical braking.

If the command string corresponds to a “forwards” or “backwards” command, a digital high corresponding to direction and an analog PWM signal with a duty cycle of 80 (out of 255) are sent to the left and right motors.

If the command string corresponds to a “lateral right” or “later left” command, a digital high and an analog PWM signal with a duty cycle of 80 (out of 255) are sent to the front and back motors. For these commands, the timeout is set to 1.2 seconds.

If the command string corresponds to a “rotate right” or “rotate left” command, then all four of the motors are commanded in the same direction using the same digital logic and PWM duty cycle as the other movements. For these commands, the timeout is set to 0.5 seconds.

Finally, if a “stop” command is received, the code writes a low value to all the motor pins to stop all drivetrain movement.

System Integration and Test

System Interfaces

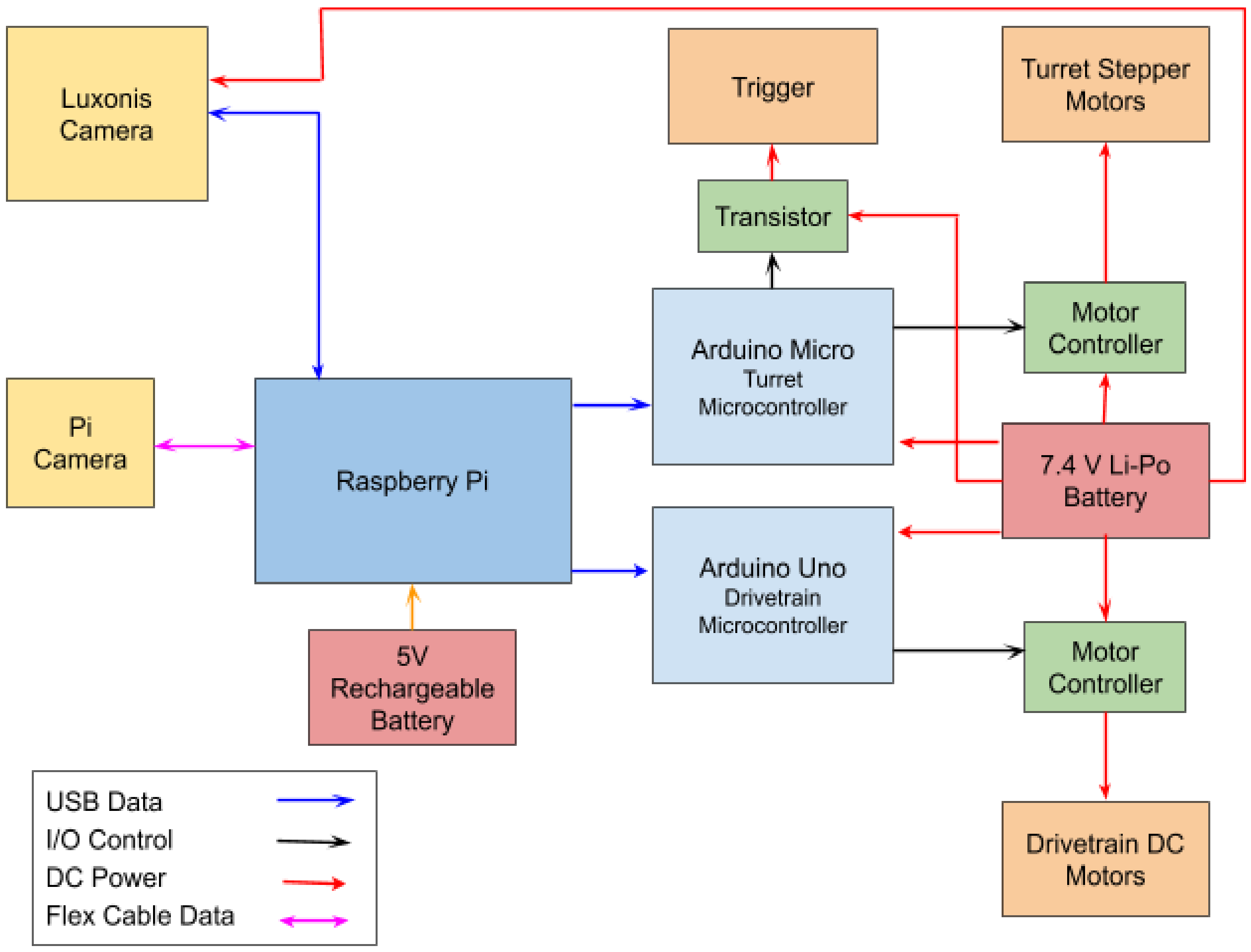


Figure 19. System Interface Diagram

Electrical Schematics

Turret Subsystem

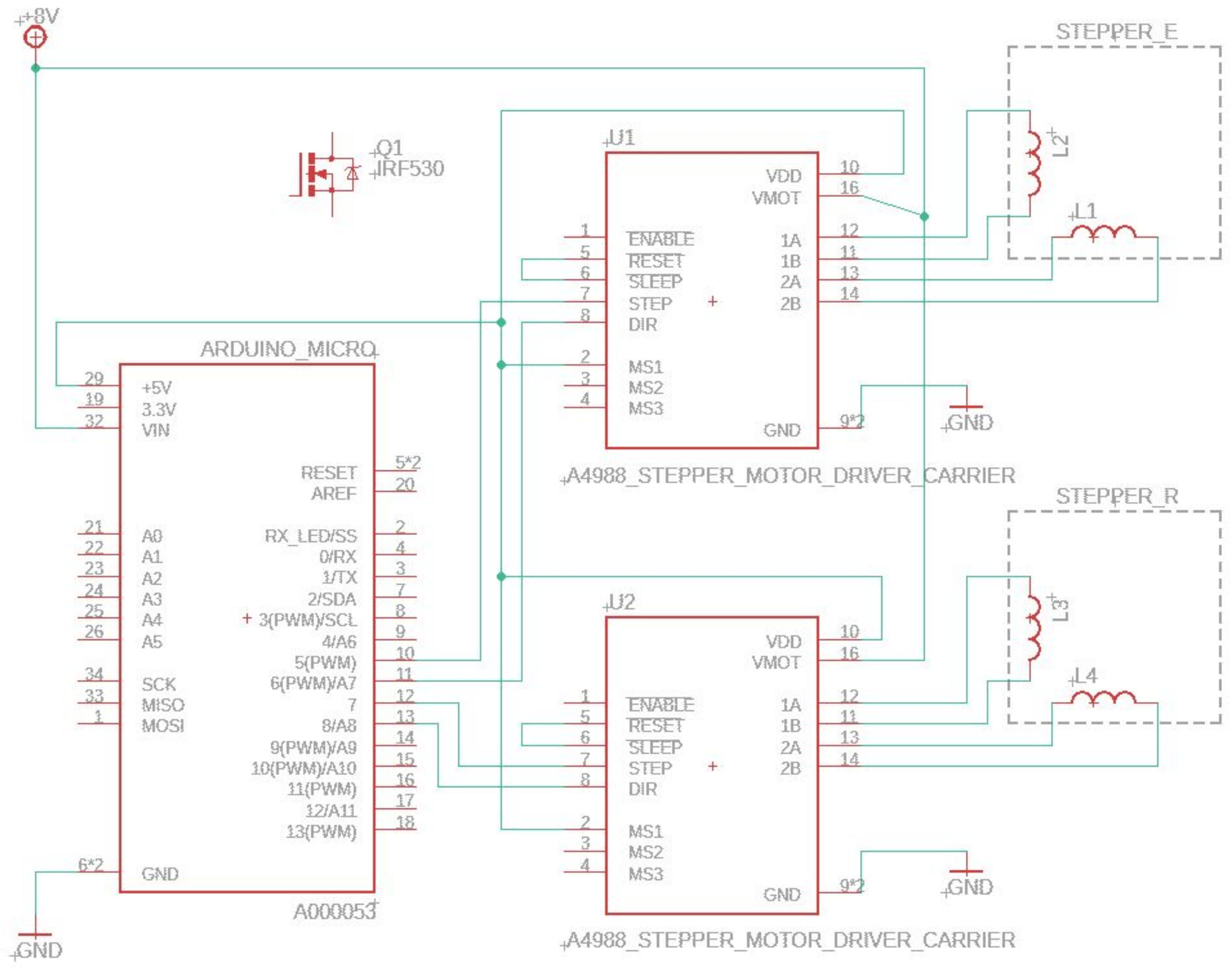


Figure 20. Turret integrated system electrical schematic

Navigation Subsystem

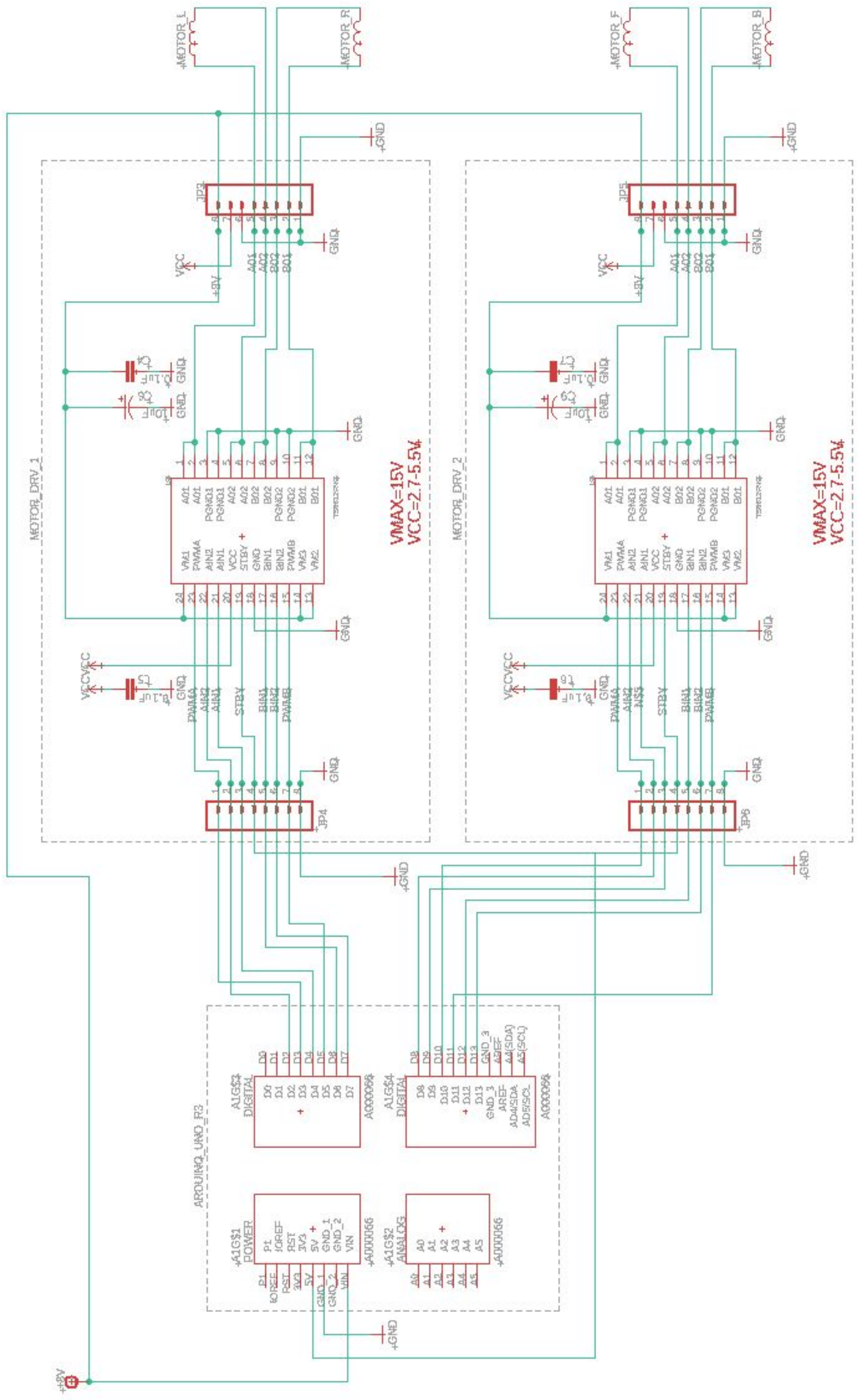


Figure 21. Navigation integrated system electrical schematic

Systems Integration

Subsystem Development Level Testing

Prior to system integration, subsystems were tested to ensure they were sufficient to meet the requirements of the system.

Structure

Requirements:
Shall have dimensions not to exceed 11” x 11” x 17” (w x w x h).
Shall have a durable structure capable of supporting and holding all components.
Testing Method:
Analysis: Structure was designed to proper dimensions
Demonstration: The materials chosen to build the structure were cheap enough that they could serve as their own prototypes. After laser cutting, the platforms were judged to have sufficient stiffness, durability, and strength for use.

Projectile Mechanism

Requirements:
Shall fire with sufficient accuracy to hit the target with a projectile.
Shall fire projectiles with sufficient force to knock over targets.
Testing Method:
Demonstration: Hand fire at project target

Power

Requirements:
Shall be internally powered with an operational power-life of no less than 15 minutes
Testing Method:
Analysis: Calculations showed that the power system was capable of meeting system requirements.
Demonstration: Components were attached to power and tested prior to full assembly to verify operation.

Drivetrain

Requirements: Shall move along two axes (linear: forward/backward, rotate: left/right) required; three axis (add lateral: left/right) preferred.
Testing Method:
Demonstration: A simple arduino script was developed to run the drivetrain through its full range of motion.

Turret

Requirements:
Shall point the projectile mechanism at red targets with sufficient accuracy to hit the target with a projectile.
Have a projectile mechanism with two axes of movement (rotate: up/down, left/right)
Testing Method:
Demonstration: A simple arduino script was developed to point the stepper motors.

Driver Software

Requirements:
Shall run Targeting and Guidance Software
Shall communicate with microcontrollers
Testing Method:
Demonstration: Testing of the driver was agile. It was debugged on a PC but primarily tested after system integration

Targeting Software

Requirements:
Shall sense, identify, and differentiate between red and green targets.
Testing Method:
Analysis: The targeting software was tested virtually on a PC
Demonstration: The targeting software was tested using a red competition target.

Guidance Software

Requirements:
Shall sense yellow and purple lines with sufficient spatial accuracy to enable turns within project constraints.
Testing Method:
Analysis: The guidance software was developed and tested using “simulated lanes” drawn in a PC paint application and then with camera images.
Demonstration: Given that the guidance software was intimately tied with the mobility subsystem and had to operate in a dynamic way, a majority of testing was done following system integration.

System Level Testing

Turret Testing

1. Calibration: The camera and projectile mechanism were calibrated (boresighted) by manually pointing the pistol barrel at the target and examining the camera’s offset (via video feed) from the center of the target. The offset values were used as parameters in the targeting software to adjust point of aim vs. point of impact.
2. Static Fire: A simple python script that commanded the pistol to fire (insta_fire.py) was developed to test communication, integration of components, and accuracy. The pistol was manually aimed and commanded to fire at the target.
3. Full System Test: The entire system was tested on a static range with guidance software disabled. This test demonstrated that the camera could locate targets, command the turret to move, center on the target, and command the pistol to fire. This process was extremely agile, requiring frequent adjustments to several components of the software in order to ensure the system functioned as required.

Guidance Testing

1. Camera Calibration: A python script was developed to test the processing of images and ensure the camera was “seeing the right things”. This was done statically by pointing the camera at the course and reviewing one image at a time.
2. Course Testing: Given that the competition course was largely unavailable for use (re: Covid-19 limitations), a mock course was developed using tape on black poster-board. The integrated guidance and mobility system was driven through different scenarios (turns, dead-ends, etc). This process was also extremely agile. The rules and architecture of the guidance code grew significantly in sophistication and complexity during this process.

End to End Testing

1. The fully integrated system was tested on the mock course to ensure the system could navigate, find targets, pause movement, shoot targets, and then resume movement throughout the course. Again, this was done over multiple iterations with frequent adjustments to software throughout.
2. Competition Course Testing/Demonstration: The robot was tested on the competition course twice, for a total duration of about 4 hours (Limited by Covid-19 restrictions).

Results

Robot Performance

The robot functioned as required and was able to navigate through portions of the course, find, identify, and shoot red targets; however, the reliability of the system as a whole to complete the entire course was limited.

Ultimately, the robot was only able to navigate around portions of the course (~4-5 turns) before making significant mistakes. Similarly, after making several turns, the robot was not always successful in knocking down red targets. Many separate and complex components of the robot (software and hardware) had a reliability of approximately 90% in controlled situations, but when used together in a single end-to-end test, a mistake or error in one component of the system caused non recoverable errors.

Given the restrictions in place due to Covid-19 quarantine, our team is happy with the demonstrated performance of the robot. The design problem proved to be far more challenging and ambitious than the team initially expected.

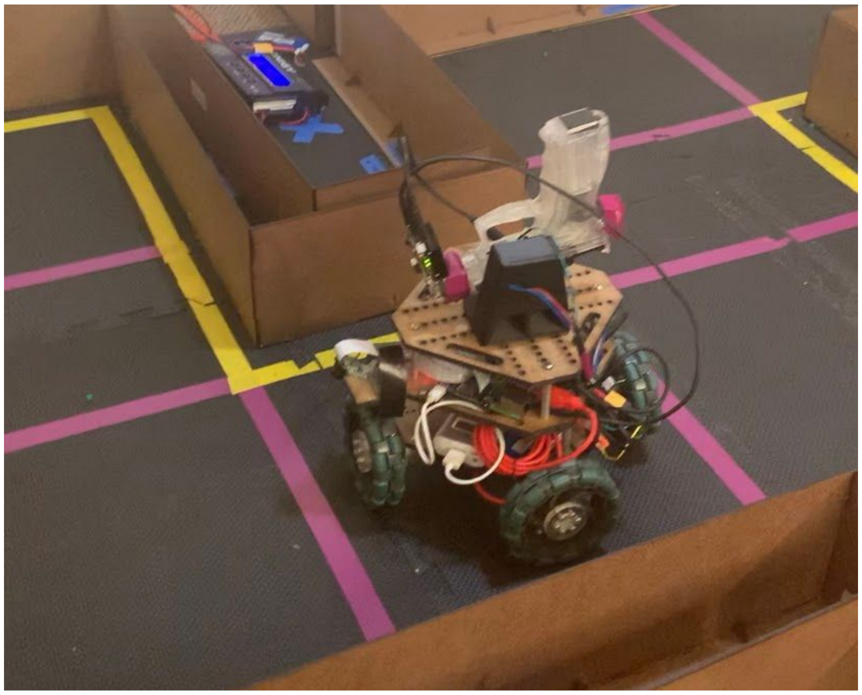


Figure 22. CLADE in operation

Major Lessons Learned

Guidance System Camera Processing Calibration: Slow and Deliberate calibration of the computer vision processing *early in the project* is extremely important. The guidance software designer initially took parameters for the processing of colors from previous examples. The processing of bright, distinct colors like yellow or light blue is relatively straightforward, but purple proved to be extremely challenging. A significant amount of time was wasted testing the robot on a course in which the robot was either over-filtering or under-filtering purple lines. The result was that the robot would either not see all the purple lines, or it would erroneously detect objects that were not purple lines (light reflections, wood floor, etc). It's critical to know exactly what your computer vision is seeing and carefully refine it until it produces limited errors.

Communication between microcontrollers: Keep it simple. Long command packets are hard to troubleshoot.

Frame: Gluing two ⅛" thick MDF boards (left image) took longer to produce (due to long wait time for super glue to dry) than using a single board of ¼" MDF. However, cutting ¼" MDF (see Figure 15 - right) takes multiple rounds of laser cutting, and tends to singe the edges. Cutting small holes is nearly impossible, and your hands get covered in soot. ⅛" thick MDF produces quick, clean cuts, and structurally feels just as sturdy as ¼" MDF, if not more due to the extra reinforcement provided by the super glue.

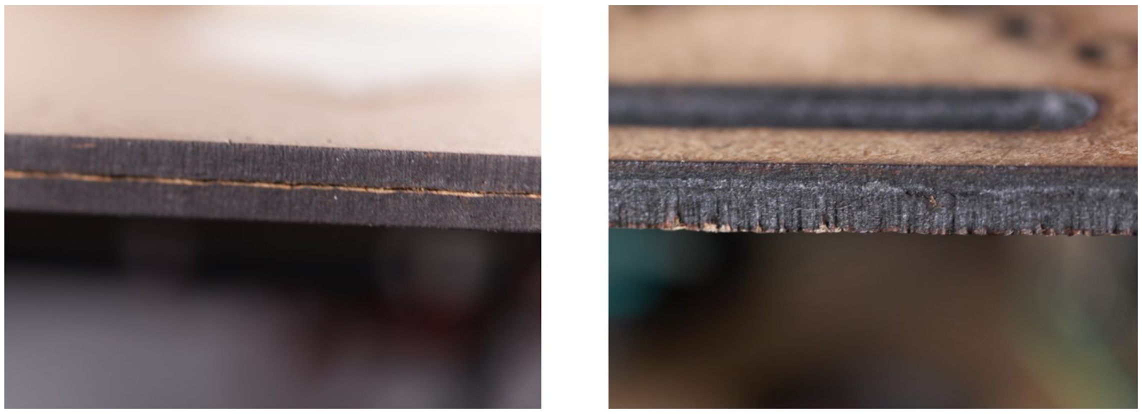


Figure 22. Comparison of (left) two ⅛" MDF boards individually laser cut with single passes and super glued, versus (right) one ¼" MDF board laser cut with multiple passes.

Power/Circuitry:

1. Use more hobby grade COTS components like Arduino motor shields or multi-motor driver PCBs. Custom wiring is always the weakest link in the designs and eats up more time than anything.
2. Having PCBs and inter-board connectors would be ideal.

Turret: Iron (Fe) weights were taped onto the barrel of the airsoft gun to counter the weight of the gun base, which served to alleviate high static torque on the stepper motor.

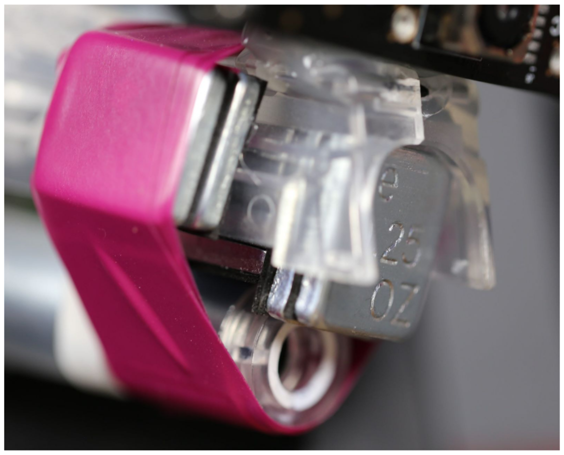


Figure 23. Pistol center of gravity weight balancing

Team dynamic/work style/communication strategies: The team communicated closely via Whatsapp to discuss issues that arose for individuals, as well as for providing updates about progress in their respective areas. Zoom meetings were also arranged in replacement of in-person meetings as a result of Covid-19 quarantine limitations to discuss goals and strategies. 'GitHub' proved essential for software development and collaboration. Using these platforms to maintain close communication was critical for maintaining good progress. The team also worked cohesively as a result of members being respectful and supportive of each other's progress while developing new skills spanning mechatronics.

Covid-19 Impacts

1. Limited access to course: Having limited access to the competition course proved to be significant. The lighting of the space, tape colors, tape reflectivity, surface material, and layout was all *slightly* different than the mock course used in testing. During the first end to end test, the team realized that all development and testing thus far had been completed on 12 inch wide lanes, and that the program was responding poorly to the 16 inch main thoroughfares. This would have been identified *much* earlier in the project otherwise.
2. Inability to meet in person: Many issues were much more challenging to solve or fix given the inability to meet in person. This was especially true of issues at the hardware to software interface. The robot often had to be handed off from one teammate to another to address issues that had arisen during testing. The result was significant delays in progress

Future Work

Machine learning: Using a rules-based approach to navigate the course with computer vision proved extremely challenging. At each step, testing continued to reveal new situations that the software could not handle or was handling improperly. As such, the developer continued to add new rules and increasingly complex nested if-then statements. In many cases, rules written to prevent an error in one situation would cause the robot to fail in another situation. Navigating intersections proved to be especially difficult - especially when they were stacked closely together on the course. This is further evident in the fact that intersection detection and navigation seems to be relatively undeveloped amongst robot hobbyists and in self driving automobiles.

While the team has limited knowledge regarding the methodology, it is possible that using a Machine Learning approach to navigate the course may prove to be more efficient and effective. In this scenario, the developer could show the robot the course, perhaps by pushing the robot through the course many times, in an effort to get the robot to develop its own rules using a machine learning algorithm. Alternatively, the developer could create a simulation course which the robot could rapidly iterate through to test itself.

Object Recognition and Tracking: The computer vision guidance software processed each image separately and treated lines independently from frame to frame. There was no object recognition or tracking built into the software, so it was not capable of knowing that the left lane it saw in frame 1 was the same left lane that it saw in frame 2, and so on. Implementing object recognition and tracking could increase the sophistication of the program and improve error correction while reducing the complexity of the rules required to function.

Additional Sensing: The team chose to use computer vision for guidance because of the learning challenge. It was incredibly rewarding and useful, but a more successful robot would most likely benefit from additional low cost sensors. Side and front mounted sonic sensors could add much needed sensing redundancy and enable the robot to better recognize when it was in the center of intersection and if it was about to strike a wall.

Supporting Documentation

Bill of Materials

Materials	Cost (approx.)	Source
MDF Board	$7	McGuckin
Standoffs/bolts/nuts/etc.	$10.00	Home Depot
Turret (3D print material)	$7.00	Home
Airsoft gun	$33.00	Amazon
Airsoft projectiles	$6.00	Amazon
Motor drivers (stepper+DC)	$15.00	AliExpress + Lab Equipment
Arduino Boards	$28.00	Amazon + AliExpress
Raspberry Pi	$35.00	Lab Equipment
Misc Electronics	$5.00	Lab Equipment
Iron Weights	$5.00	O'Reilly's Auto Parts
Pi-Camera Wide angle Lens	$25.00	Amazon
Total	**$176.00**	

Source Code

Directory:
https://github.com/KineticPayload/MCEN5115

Driver (Operating System) Software:
https://github.com/KineticPayload/MCEN5115/blob/master/driver_v2.py

Targeting Software
https://github.com/KineticPayload/MCEN5115/blob/master/targeting.py

Guidance Software:
https://github.com/KineticPayload/MCEN5115/blob/master/LaneGuidev10.py

Motor Driver
https://github.com/KineticPayload/MCEN5115/blob/master/MotorDriver/MotorDriver.ino

Turret Driver
https://github.com/KineticPayload/MCEN5115/blob/master/Turret/Turret/Turret.ino

References

Tian, David. (2019, April). DeepPiCar: How to Build a Deep Learning, Self Driving Robotic Car on a Shoestring Budget. https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c