A group of boys wearing matching shirts

AI-generated content may be incorrect.

***Team Hyperion***

**2026 Planning**

|  |  |
| --- | --- |
| **Hardware** | **Software** |
| Matthew Adams, Electrical | Sam Garg, Movement & Strategy |
| Luke Atherton, Structural | Thomas McCabe, Sensing & Control |

Contents

[Software 4](#_Toc212396267)

[Assignments 4](#_Toc212396268)

[General Conventions 4](#_Toc212396269)

[1. TSSP System – Thomas McCabe 5](#_Toc212396270)

[Define Problem 5](#_Toc212396271)

[Background Research 5](#_Toc212396272)

[Requirements 5](#_Toc212396273)

[Plan for Solution 5](#_Toc212396274)

[2. Movement Strategy – Sam Garg 5](#_Toc212396275)

[Attacker 5](#_Toc212396276)

[Defender 7](#_Toc212396277)

[3. Light System – Tom McCabe 9](#_Toc212396278)

[Define Problem 9](#_Toc212396279)

[Background Research 9](#_Toc212396280)

[Requirements 9](#_Toc212396281)

[Plan for Solution 10](#_Toc212396282)

[4. Camera Library (OpenMV Side) – Tom McCabe 10](#_Toc212396283)

[Define Problem 10](#_Toc212396284)

[Background Research 10](#_Toc212396285)

[Requirements 10](#_Toc212396286)

[Plan for Solution 10](#_Toc212396287)

[5. Camera Library (Teensy Side) – Sam Garg 10](#_Toc212396288)

[Define Problem 10](#_Toc212396289)

[Background Research 10](#_Toc212396290)

[Requirements 10](#_Toc212396291)

[Plan for Solution 10](#_Toc212396292)

[6. Bluetooth – Sam Garg 11](#_Toc212396293)

[Strategy 11](#_Toc212396294)

[Sending and Receiving 15](#_Toc212396295)

[7. Kicker Mechanics – Sam Garg 19](#_Toc212396296)

[Define Problem 19](#_Toc212396297)

[Background Research 19](#_Toc212396298)

[Requirements 19](#_Toc212396299)

[Plan for Solution 19](#_Toc212396300)

[8. Kicker Strategy – Tom McCabe 20](#_Toc212396301)

[Define Problem 20](#_Toc212396302)

[Background Research 20](#_Toc212396303)

[Requirements 20](#_Toc212396304)

[Plan for Solution 20](#_Toc212396305)

[9. Dribbler Mechanics – Tom McCabe 20](#_Toc212396306)

[Define Problem 20](#_Toc212396307)

[Background Research 20](#_Toc212396308)

[Requirements 20](#_Toc212396309)

[Plan for Solution 21](#_Toc212396310)

[10. Dribbler Strategy – Sam Garg 21](#_Toc212396311)

[Define Problem 21](#_Toc212396312)

[Background Research 21](#_Toc212396313)

[Requirements 21](#_Toc212396314)

[Plan for Solution 21](#_Toc212396315)

[11. RGB LED Mechanics – Tom McCabe 21](#_Toc212396316)

[Define Problem 21](#_Toc212396317)

[Background Research 21](#_Toc212396318)

[Requirements 21](#_Toc212396319)

[Plan for Solution 21](#_Toc212396320)

[12. RGB LED Code Debugging – Sam Garg 21](#_Toc212396321)

[Define Problem 21](#_Toc212396322)

[Background Research 21](#_Toc212396323)

[Requirements 22](#_Toc212396324)

[Plan for Solution 22](#_Toc212396325)

[Hardware 23](#_Toc212396326)

[Problems Identified in 2025 23](#_Toc212396327)

[Solutions to problems 23](#_Toc212396328)

[Brainstorm Design Ideas 24](#_Toc212396329)

[Structural 24](#_Toc212396330)

[Electrical 25](#_Toc212396331)

[List of References 27](#_Toc212396332)

# Software

## Assignments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aspects | Assigned To | Individual Order | Overall Order | Status |
| Bluetooth | Sam | 3 | 6 | Complete  (26/10/25) |
| Camera (Teensy Side) | Sam | 2 | 5 | Awaiting OpenMV Side |
| Camera (OpenMV Side) | Tom | 3 | 4 |  |
| Drive System | N/A (Prev.) | N/A (Prev.) | N/A (Prev.) | Complete  (Prev.) |
| Light System | Tom | 2 | 3 |  |
| PID | N/A (Prev.) | N/A (Prev.) | N/A (Prev.) | Complete  (Prev.) |
| TSSP System | Tom | 1 | 1 | Complete  (23/10/25) |
| Timer | N/A (Prev.) | N/A (Prev.) | N/A (Prev.) | Complete  (Prev.) |
| Voltage Divider(s) | N/A (Prev.) | N/A (Prev.) | N/A (Prev.) | Complete  (Prev.) |
| Compass | N/A (Prev.) | N/A (Prev.) | N/A (Prev.) | Complete  (Prev.) |
| Kicker Mechanics | Sam | 4 | 7 | Complete  (26/10/25) |
| Dribbler Mechanics | Tom | 5 | 9 |  |
| Kicker Strategy | Tom | 4 | 8 |  |
| Dribbler Strategy | Sam | 5 | 10 |  |
| RGB LED Mechanics | Tom | 6 | 11 |  |
| RGB LED Code Debugging | Sam | 6 | 12 |  |
| Movement Strategy | Sam | 1 | 2 | Complete  (21/10/25) |
| Common | Shared | Shared | Shared | Shared |
| Configuration | Shared | Shared | Shared | Shared |
| Pins | Shared | Shared | Shared | Shared |

## General Conventions

Variables – Camel Case

Class Names – Camel Case with a capital starting letter

Function and Struct Names – lower case separated by underscores

## 1. TSSP System – Thomas McCabe

### Define Problem

To be able to get possession of the ball and score, the direction of the ball must be found.

### Background Research

There are multiple ways to detect and determent the relative direction of the ball. Given the ball is orange a camera such as an open mv h7 can track the colour to determine the balls position, the downside of this being the frame rate of the camera and the ball not being visible for the camera. The ball that is used in comp emits infrared light with can be detected by infrared sensors, as infrared light is light it is a lot harder to be blocked making it more reliable.

### Requirements

Using 24 infrared sensors the robot can detect the ball as well as the direction of the ball relative to the robot.

### Plan for Solution

The plan is to read each ir sensor 255 times adding their digital value of 0 or 1 to their total value that the direction of the ball is dictated based in the direction of the highest valued sensor in degrees.

Functions

|  |  |
| --- | --- |
| Function Name | Function |
| init\_tssp | To set up the pins and their types |
| read\_tssp | To read all the tssp and find ball dir and str |
| get\_ball\_dir | Returns ball dir |
| get\_ball\_str | Returns ball str |

## 2. Movement Strategy – Sam Garg

### Attacker

#### Define Problem

To not constantly hit the ball towards the defending goal, the robot must have a movement strategy to effectively get behind the ball and move it towards the opponent’s goal.

**What kind of attacking movement strategy should the robot have?**

#### Background Research

There are many different types of orbits/movement strategies, which can include:

|  |  |  |  |
| --- | --- | --- | --- |
| Orbit Type | What is it? | Positives | Negatives |
| Cases Orbit | A case's orbit is a set movement angle when the ball angle is within a certain range of angles. | * Easy to set up. * Predictable movement (always behave the same within each case). * Low processing demand. | * Lengthy and tedious tuning. * Jerky movement. * Does not adapt well to unexpected ball positions or fast movement. * Hard to maintain smooth transitions between cases. |
| Fixed Offset Orbit | A fixed offset orbit is a set constant that adds or subtracts from the ball's direction. | * Easy to set up. * Easy to tune. * Consistent and simple. * Low processing demand. | * Orbit can be very wide or too close at times, with no variance. * Lack’s flexibility * Struggles when the ball moves quickly or unpredictably. |
| Exponential  Polynomial Orbit | An exponential/polynomial orbit is a constantly changing offset value that is subtracted or added to the ball’s direction. | * Smooth movement. * Tightens naturally near the ball. * Looks natural and efficient in play-like situations. * Handles varied speed sand angles better than fixed systems. | * Complex to set up. * Harder to predict behaviour/interpret * Medium processing demand. |
| PID Orbit | A PID Orbit has a vertical and horizontal vector; it uses these vectors to wrap around the ball. | * Very smooth and adaptive movement around the ball * Continuously fixes itself based on readings; maintains stability. * Adjustable responsiveness. It can be very aggressive if needed. | * Very complex to set up. * Very complex to tune. * Can overcorrect or oscillate if not tuned properly. * Very high processing demand. * Relies on ball strength and distance; cannot function without. |

#### Requirements

The orbit movement strategy must:

* Be easy to tune and adjust.
* Position the robot efficiently behind the ball.
* Prevent unwanted contact with the ball (from sides or rear)
* Adapt to different ball speeds and movement patterns.
* How low to medium processing demand (to optimise the loop speed)
* Be stable and predictable
* React quickly to sensor data
* Maintain smooth curvature in its orbit path
* Allow for scalable complexity (make it more advanced)
* Minimize overshooting

#### Plan for Solution

Based on the background research and the outlined requirements, an **Exponential Orbit** strategy will be implemented in the program. This method will rely on a single constant to tune the overall orbit behaviour.

1. A **modified ball direction** will be calculated, ranging from -180° to 180°. This allows both sides to be mirrored, ensuring the robot produces symmetrical movement on either side of the ball.
2. A **movement scalar** will be determined based on the ball’s position. This value will range between 0 and 1 and follow an exponential curve, allowing for smoother and more responsive movement as the ball gets closer.
3. A **movement offset** will be generated, increasing exponentially based on how far off-centre the ball is. This creates a smooth orbiting path, with the offset capped at 90° to prevent the robot from turning completely sideways. The offset will then be added or subtracted from the ball’s direction, depending on which side the ball is on, to determine the final movement direction.

This plan will be integrated into the TSSP library as a separate function named orbit().

### Defender

#### Define Problem

The opponents’ robots will aim to score against our goal. To stop them from scoring when the attacker cannot get back around the ball in time, it is wise to have a robot hang back, defending the goal.

**How should the Defender move, and what algorithms can be used to calculate the Defender's movement?**

#### Background Research

Defending can be done in many ways. The following table describes the different methods that can be used to program the defender.

|  |  |  |  |
| --- | --- | --- | --- |
| Method | What is it? | Positives | Negatives |
| Distance Constrained Defence | The robot orbits around the ball but stops moving forward once it reaches a set distance from the goal. | * Not commonly seen and provides a level of difference * Higher chance of scoring compared to staying in deep defence * Helps maintain goal positioning between the ball and goal. | * If opponents have a strong kicker, the robot may not orbit back fast enough to block the ball. * Can leave the goal open if timing or distance calibration is off. |
| Reactive Zone Defence | The robot stays between the goal and the centre of the field, moving or orbiting only when the ball enters its defensive zone. | * Conserves energy and reduces unnecessary movement. * Provides reliable defensive coverage when the ball approaches. * Easier to tune than complex systems. | * May take too long to react to fast-moving balls. * Can struggle against opponents with quick gameplay. |
| Floating Defender using Vectors | The robot keeps a constant distance from both the ball and the goal, always positioning itself between them to be ready for quick attacks or strong defence. | * Offers balanced positioning for both offence and defence. * Enables faster transitions between attacking and defending. * Maintains optimal alignment with the ball and goal. | * Requires precise vector calculations and tuning. * Can become unstable if the ball moves unpredictably and it is not tuned correctly. |
| Defender using Vectors | The robot moves sideways with the ball while saying a set distance away from the goal. | * Simple and stable defensive approach * Effective for cutting off shots to the side of the goal with goal tracking. * Easier to implement and maintain. | * Less responsive to central attacks. * Doesn’t adapt to ball position relative to the goal (distance wise). * Can create open angles for shots if not tuned properly. |

#### Requirements

* Maintain Goal Coverage
* Efficient Movement (minimise unnecessary movement)
* Fast Reaction Time
* Stable Positioning
* Accurate Distance Control
* Smooth & Visible Switching (You must be able to see a visible change between attacker and defender strategies).
* Algorithm Flexibility & Tuning

#### Plan for Solution

Given the positives and negatives of the chosen solutions, a floating defender using vectors will be picked. This is because the robot will be positioned between the ball and the goal, which maintains coverage. It can be easily tuned to minimise unnecessary movement and can be stable when velocities are limited, and dampening is added.

* When the ball is not on the field, the robot will centre relative to the goal the minimum distance away (this will be a define to tune).
* When the ball is on the field, the robot will use a tuned horizontal vector to position itself in front of the ball.
* The robot will actively face away from the goal so that when the robot moves towards the ball it faces between the ball and goal.
* A maximum and minimum goal distance will be defined to ensure that the robot does not move too excessively towards or away from the defending goal.
* A scaling constant will be defined between the ball strength and the goal distance, attempting to make them similar values. Whichever value is smaller will be multiplied by the scaling constant.

The logic for this solution will be placed inside the main file as it requires multiple objects.

## 3. Light System – Tom McCabe

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 4. Camera Library (OpenMV Side) – Tom McCabe

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 5. Camera Library (Teensy Side) – Sam Garg

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 6. Bluetooth – Sam Garg

### Strategy

#### Define Problem

To not crowd the ball and to split roles on the field, the robot will need to have a strategy as to deciding which one is the attacker, and which one is the defender.

**What is the condition(s) for deciding which robot is attacking and which is defending?**

#### Background Research

The following table describes the different conditions/methods that can be used to decide the attacker/defender roles in a game situation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | What is it? | Positives | Negatives | Combine with |
| Strongest Ball Strength | The robot with the strongest ball strength becomes the attacker. The other becomes the defender by default. | * Simple and reliable indicator of which robot is closest to the ball. * Fast decision-making with minimal data processing * Reduces confusion in determining possession. * Works without more complex components (e.g. Camera). | * Can cause rapid switching if both robots have similar ball readings. * Doesn’t consider field position (may send both robots too far forward). * Could result in collisions near the ball if not paired with direction/goal checks. | Ball Direction Attack Cone.  Distance-from-own-goal Fallback |
| Ball Direction Attack Cone | If a robot’s shows the ball within a forward cone (e.g. ±30 degrees of its heading) and the ball strength is similar to that of the other robot, the robot should be the attacker. The other robot becomes the defender by default. | * Ensures the attacker is properly oriented toward the ball. * Reduces risk of sideways or backwards approaches. * Promotes smoother, more strategic ball control and chasing behaviour. * Avoids conflicts where both robots see the ball from opposite sides. | * Relies on more accurate heading readings, errors can cause false alignment. * May ignore a closer robot if its angle is slightly off. * Slower to decide if angle calculations are noisy. | Strongest Ball Strength  Goal Alignment |
| Goal Alignment | If a robot can see the opponent goal and the goal heading is generally forward from it, prefer that robot as attacker. The other becomes the defender by default. | * Encourages intelligent play by choosing robots that are well-positioned to score. * Prevents attackers moving in the wrong direction. * Improves orientation consistency. * Increases scoring efficiency once ball possession is achieved. | * Goal detection can be inconsistent under lighting changes. * May fail if the goal is blocked or temporarily useless. * Doesn’t help much during defence phases or when ball is near your own goal. | Ball Direction Attack Cone  Distance-from-own-goal Fallback |
| Distance-from-own-goal Fallback | Whichever robot is closer to the goal defaults to the defender. If both robots are near the defending goal, use other conditions (such as strongest ball strength) to determine roles. | * Ensures at least one robot is always defending the goal. * Prevents both robots from rushing too far up the field. * Provides structure and balanced team positioning. * Easy to determine using compass and estimated positioning using camera. | * Position may be slightly inaccurate/inconsistent at times. * May cause excessive hesitation if both robots stay defensive. * Can conflict with goal alignment if the “defender” also has a strong shot. | Strongest Ball Strength  Goal Alignment |
| Battery Priority | If one robot’s battery/temperature/health is below a safe threshold, deprioritize it for attacking tasks. | * Reduces risk of inconsistent sensor values or slowed gameplay on one robot compared to the other. * Reduces heat buildup and motor strain on weaker robot. * Prevents performance drops during key moments and scoring opportunities. | * Requires constant battery monitoring and sending this via Bluetooth. * Can cause uneven role assignments if one robot’s battery remains low for the entire match. * Doesn’t consider game context (e.g. what is happening in the game) | Strongest Ball Strength  Distance-from-own-goal Fallback |

#### Requirements

* Consistent Role Assignment Logic
* Scalability
* Simplicity
* Small Number of Aspects/Values Required (to prevent the number of things being sent over Bluetooth)
* Synchronization (One robot must be attacking and one must be always defending)

#### Plan for Solution

A variety of the above brainstormed strategies will be incorporated into the final solution.

The robots will share key data values:

* Ball Strength
* Ball Direction
* Goal Angle (Attacking Goal)
* Goal Distance (Defending Goal)
* Battery Level

Using these values, the robot will decide whether it is an attacker or defender based on prioritised on the following set of conditions in a hierarchy.

1. **Compare Battery Levels**

If one robot’s battery is low, it automatically becomes the **defender**.

1. **Compare Ball Strength**

The robot with the higher ball strength becomes the **attacker**, unless Step 3 or 4 overrides it.

1. **Check Ball Direction (Attack Cone)**

If one robot sees the ball within a forward cone (±30° of its heading) and the other doesn’t, that robot becomes the **attacker**.

1. **Goal Alignment**

If a robot is oriented toward the opponent’s goal, it remains or becomes the **attacker**.

1. **Use distance-from-own-goal fallback**

If both robots meet similar attack criteria, the one closer to the defending goal becomes the **defender**.

With these conditions, the behaviour of each robot in each role should appear as:

* **Attacker:** Moves toward the ball, maintains alignment with the opponent’s goal, and attempts to shoot when close enough.
* **Defender:** Holds position closer to its own goal, tracking both he ball and the attacker’s location to intercept or block if the ball approaches the goal area.

In addition to this, to ensure that there is a “fail-safe” to this logic, should Bluetooth data be disconnected, lost, or unclear, roles default to **Defender** logic. Having two robots in the back half of the field is better than leaving the goal open and exposed.

### Sending and Receiving

#### Define Problem

To send the values that decide strategies across both robots, a process must be defined.

**What process will this be?**

#### Background Research

**How receiving should work:** (Source 1 of this Document)

1. First check if there is anything that is available on the serial (Serial.available()). If there is anything available, continue to the next step, if not keep checking.
2. Serial.read() to get this value and delete it from the frame buffer.

This is how values can be received from the other HC-05 Bluetooth module.

**What we can take away from this source:** How to send and receive using HC-05 Bluetooth module.

**Noise across serial lines:** (Source 2 of this Document)

In the above source, we can see that the user is experiencing issues with the HC-05 Bluetooth module sending odd values here and there. This is called noise. To prevent noise, two start bytes can be used to ensure that the values being read are inside the packet that we want to read.

E.g. Start bytes can be 255 and 255 in a row as no sensor input will output 255 twice in a row.

**What we can take away from this source:** We should include start bytes in our sending and receiving codes to find the start of a packet of values.

**Sending:** (Source 3 of this Document)

In the above source, we can see how to send values from a HC-05. This can be a simple Serial.write() function to the specific serial. Ensure the 2 start bytes are sent first before any other values are sent.

**HC-05 Packet Support:** (Source 4 of this Document)

The HC-05 can support up to 38400 bits per second (38400 baud rate).

|  |  |
| --- | --- |
| Name | Minimum Required Data Size |
| Byte1 | 1 byte (8 bits) |
| Byte2 | 1 byte (8 bits) |
| InfoByte | 1 byte (2 bits) |
| Ball Strength | 1 byte (8 bits) |
| Ball Direction | 2 bytes (9 bits) |
| Goal Angle | 2 bytes (9 bits) |
| Goal Distance | 1 byte (8 bits) |
| Battery Level | 1 byte (8 bits) |

Therefore, the data size of the packet will be a minimum of **9 bytes**.

Calculate the packet size:

Calculate the maximum number of times it can send per second:

Calculate how frequent this is (microseconds):

For the HC-05 to be running at maximum capacity, we should send our packet every 2.083 milliseconds. However, to not overload the HC-05 and teensy module, we should run the HC-05 at a maximum of 10% of its performance. Therefore,

Algorithm based on performance:

**What we can take away from this source:** For the HC-05 to be running at 10% capacity, a packet should be sent every 20.83 milliseconds.

#### Requirements

* Accurate Data Transmission
* Start Byte Packet Framing
* Error Resistance and Noise Handling (Ignore any data received before the start bytes)
* Defined Packet Structure
* Efficient Transmission Timing (Controlled Intervals)
* Fail-safe Defaults
* Compact Data Format (Compressed Values)

#### Plan for Solution

To ensure consistent and reliable Bluetooth communication between both robots, the following sending and receiving system will be implemented.

The defined data packet structure will be as follows:

* [Byte1, Byte2, InfoByte, BallStrength, BallDirection, GoalAngle, GoalDistance, BatteryLevel]

As discovered previously, this packet will be sent every 20.33 milliseconds to run the HC-05 at 10% performance. However, this can be tested and tuned using the function:

There will be a few main functions inside the library, the following table describes these.

|  |  |  |
| --- | --- | --- |
| Function | Brief Explanation | Plan |
| Init (void) | Prepares the library for usage in the setup. | * Begin serial communication at the defined baud rate. (38400) * Reset and start all timers. |
| update (void) | Updates the whole object in a loop. | * Update local self-data with current sensor readings. * Call send() when the timer expires and read whenever data is available. * Determine if the Bluetooth connection is alive using a timer. * Use hierarchal decision logic to assign self.role and manage switching (the details can be seen in the strategy section for Bluetooth). * Prevent Rapid oscillations with a cooldown timer. * Handle fail-safe defaults.   + If Bluetooth disconnects 🡪 force defender.   + If both robots can’t see ball 🡪 pause switching. * Include debug output for testing. |
| send (void) | Writes the data to the other HC-05 module. | * Send two start bytes (255, 255) * Send all fields in compressed format (defined above in the data packet structure) * Converts floats into scaled integers for consistent transmission. * Reset a timer after transmission. |
| read (void) | Receives the data from the other HC-05 module. | * Continuously check if enough bytes are available to form a full packet * Look for two consecutive start bytes (255, 255) * Once found, read remaining bytes in defined order and decode them into other data fields. * Update a timer each time a valid packet is received |
| get\_role (Get Event) (bool) | Outputs the role of the local robot. | * Return the Boolean value of self.role * No calculations occur here. |

## 7. Kicker Mechanics – Sam Garg

### Define Problem

To use our kicker, we must create a function which allows us to ‘fire’ the kicker. There are some constraints as to when we should fire our kicker, and how this can be done easily through the code.

**What are the constraints for the kicker software wise, and how can this be implemented via code?**

### Background Research

**Kicking too frequently:**

* Teams in the past have reported kicking too frequently. To fix this, we will implement a timer which will ensure that the kicker does not fire too frequently.

**Kicking before enough voltage is generated:**

* To ensure that the kicker does not fire before enough voltage has been generated, a voltage divider will be read using the voltage divider library and then passed into a function.

**How to Kick:**

* Writing high to a digital pin which enables power to the kicker.

### Requirements

* The kicker must only fire when the capacitor voltage is above a defined threshold.
* The kicker must have a cooldown timer to prevent it from firing too frequently.
* The kicker must activate through a controlled digital output (e.g. writing HIGH to a specific pin for a short, defined pulse duration)
* The kicker logic must be efficient and non-blocking (no delay() calls that pause other functions)
* The software must protect against accidental double-triggering caused by loop timing or communication errors.

### Plan for Solution

When combining the above information, a clear plan for the solution can be found. Below are the two functions that will be inside the library:

|  |  |  |
| --- | --- | --- |
| Function | Parameters | Functionality |
| init (void) | None. | Initialises the library for use:   * Define the pin mode for the kicker pins. * Reset timers. |
| fire (void) | Capacitor Voltage (voltage divider) | * Continuously read the capacitor voltage using the voltage divider library (value fed into function) * Store the voltage value and compare it against a defined threshold to ensure efficient charge. * Use a timer system to track cooldown time between kicks and prevent frequent firing. * When voltage and timer conditions are met, briefly write HIGH to the kicker’s digital output pin to activate the solenoid. * After a short, defined pulse (100-200 ms), set the pin LOW to deactivate the kicker. * Reset the cooldown timer immediately after each kick to manage timing between activations. |

## 8. Kicker Strategy – Tom McCabe

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 9. Dribbler Mechanics – Tom McCabe

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 10. Dribbler Strategy – Sam Garg

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 11. RGB LED Mechanics – Tom McCabe

### Define Problem

A

### Background Research

A

### Requirements

A

### Plan for Solution

A

## 12. RGB LED Code Debugging – Sam Garg

### Define Problem

A

### Background Research

A

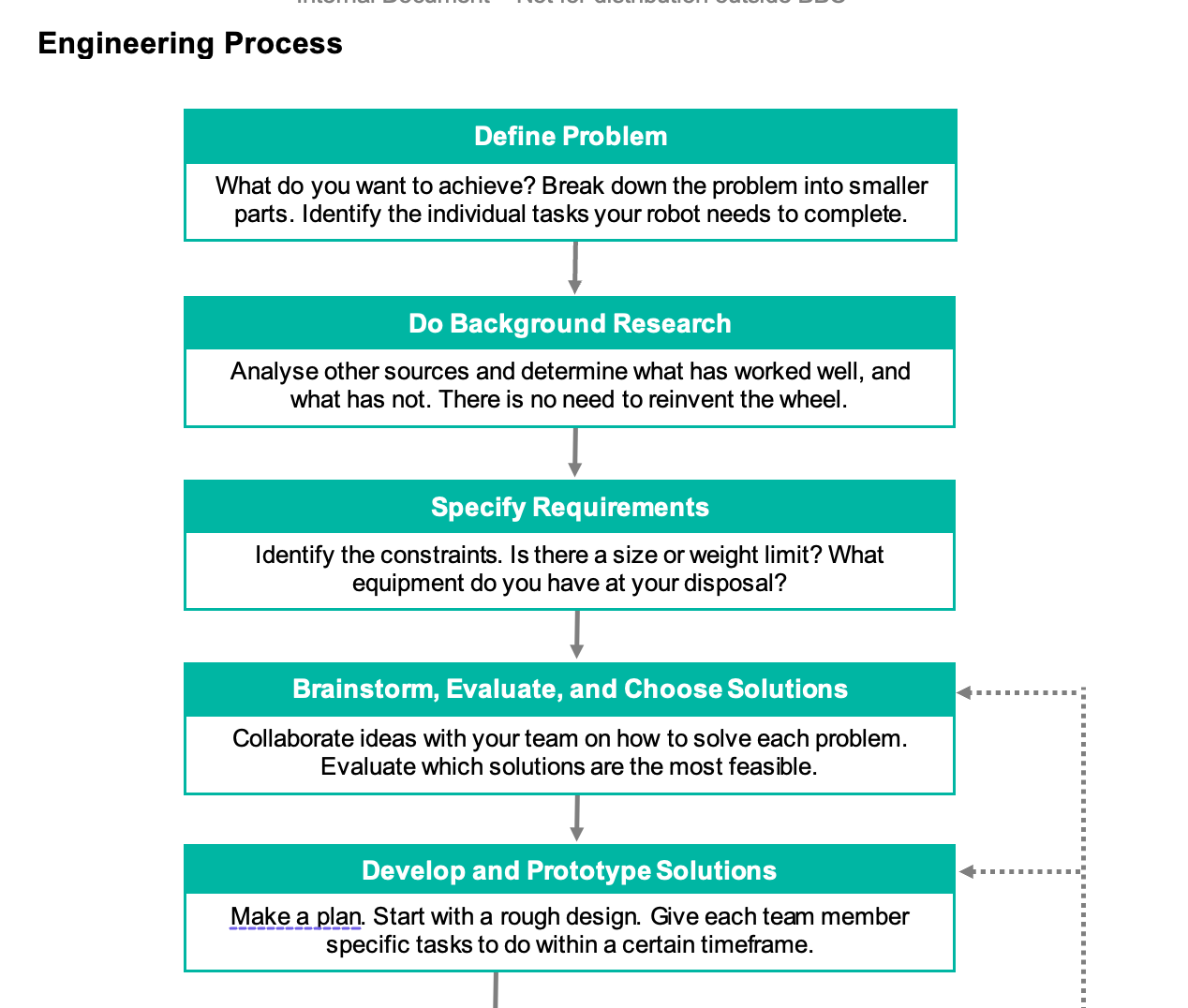
### Requirements

A

### Plan for Solution

A

# Hardware



## Problems Identified in 2025

1. Wheels didn't have enough grip when robots ran at high speeds when surging, therefore, they slid on the field.
2. Battery position and replicability were inefficient and not safe
3. The top of the Main Board was inaccessible without taking off the adjustable tube/mirror mount
4. The bottom of the Main board was inaccessible without taking apart the robot
5. The light sensor board scraped against the painted white lines at States
6. The light sensor board could have shorted when positioned against the aluminum base plate
7. The international communication module was not considered in the design before ordering/printing

## Solutions to problems

1. A double-layered wheel should be used for more grip on the field due to its increased surface contact area with the ground.
2. The battery connector extension should extend out more, and the battery position should be
3. The size of the adjustable tube/mirror mount should be decreased to not cover the entire main board.
4. Adjust how the TSSP ring is mounted to the robot. Instead of having the TSSP ring standoffs screw into the mid plate and the motor holder

## Brainstorm Design Ideas

### Structural

#### Frame

* 220mm diameter limit therefore our robot will be 210mm
* Aluminium 6061 for endurance due to 3D printed PLA being flimsy and brittle

#### Movement

* 4x DCX 19 Maxon Motors in X shape
* Double omni wheels for extra grip at high speeds with thin guard around them for protection
* 

#### Vision System

* Acrylic tube for a full 360 view of the field with compact yet accessible adjustment plates to Centre mirror
* Camera and hyperbolic mirror for goal tracking and orange ball detection
* Hyperbolic mirror (track ball and goals, potentially also for coordinate system)

#### Ball Capture

* Curved capture zone for more contact area
* <https://junior.robocup.org/rcj-ir-soccer-ball-2026-changes-announced/> (42mm diameter ball, use ir and camera to track) (‘ball-capturing zone of up to 1.5 cm’)
* Spherical cut out, with space for kicker)

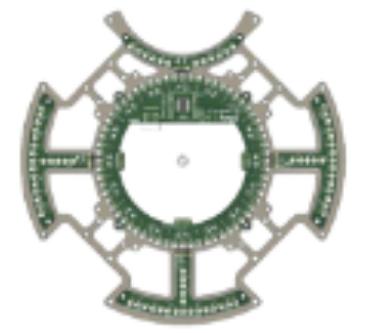


### Electrical

#### Main Board

* 24x TSSP58038’s for ball detection on 1x Teensy 4.0 for more accurate strength and direction readings
* 24 RGB LED’s matching TSSP’s (don't have to use them, but it's for the funnies, and we can use it to showcase the ball direction, also can be used to detect the different roles of the robots to see if Bluetooth is connected and when it switches) (PLEASE LET US DO THIS)
* HC-05 Bluetooth modules for switching strategy
* BNO005 for compass correction
* OpenMVH7-plus for orange ball detection, goal tracking and localisation
* DCX19 Maxon motor for dribbler motor
* 4x VNH7070’s motor controllers
* Teensy 4.1 as main micro controller
* 5V and 3.3V regulators for logic line
* Voltage divider to read battery voltage
* 1x switch for discharging capacitors
* 1x switch for robot power on/off
* 1x switch for logic power on/off
* 1x switch for RGB on/off (if we have it) (AGAIN PLEASE LET US ADD RGB)
* 1x switch for dribbler enable/disable
* XT60 extension for battery connection
* FFC down to LSB

#### Light Sensor Board

* 2 rings light sensor board [Example:]

(either 32, 48 or 64 LED’s and no straight parts aiming towards the middle)

For fast line detection, space with wheels going to ward the edge of the robot so there is room for the solenoid, and inner ring for defender.

#### Kicker Board

* 1x 10000uf capacitor connecter for storing voltage for solenoid
* 1x Solenoid connecter for kicking the ball
* Voltage divider for capacitor reading.

# List of References

1. <https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/>

2. <https://forum.arduino.cc/t/hc-05-bluetooth-module-interference-with-heart-beat-sensor/637639>

3. <https://forum.arduino.cc/t/sending-a-value-between-arduinos-using-hc-05-bluetooth-modules/859376>

4. <https://www.electronicwings.com/sensors-modules/bluetooth-module-hc-05->