B4W_BB8: A Spherical Self-Propelling Smart System for Monitoring and Surveilling Remote Areas

Hugo van Dijk (4864921), Remy Duijsens (4448766)

***Abstract*—This paper introduces Balls for Walls: BB8 (B4W BB8), a spherical self-propelling smart system. Unlike traditional static sensor nodes, B4W BB8 is equipped with self-propelling capabilities that allow it to position itself at specific locations. The device uses gyroscope, magnetometer and accelerometer data to calculate its position via a localization algorithm. With a compact design of 8cm diameter, B4W BB8 is expandable, providing flexibility for integrating various sensors. Several experiments are conducted to verify the performance of the system, such as angle reading verification, straight line divergence, navigation accuracy, controllability, inclined surface test and power consumption analysis. The findings expose difficulties in accurate self-navigation but demonstrate that the system can be successfully operated remotely. Prospects for future research consist of exploring alternative displacement measurement means, repositioning the systems' centre of gravity and investigating commercially available motor-gear systems.**

I. Introduction

Remote areas like Chernobyl or the Sahara desert are difficult to monitor by humans due to their extreme conditions. Authorities might want to detect possible intruders, and biologists might want to detect wildlife presence in these kinds of areas. Sharma et al. [1] proposed *Balls for Walls* (B4W), an alternative to traditional fences consisting of multiple ball-shaped sensor nodes to detect intruders. Their design included three microphones for detection and was designed to be deployed into an area with a specialized launcher. However, once a wall has been formed, it will stay in that orientation forever. The sensor nodes lack one major feature: self-propulsion. Self-propulsion would allow the balls to position themselves in a more optimal location, or rearrange when necessary. To dive into the above feature addition, we present *Balls for Walls: BB8* (**B4W_BB8**), a spherical self-propelling smart system for monitoring and surveilling remote areas. Our proposed system proves a mobile, ball-shaped system can navigate itself to set targets while allowing expandability with other sensors. All while being packaged in small (8cm) shells. The system incorporates a localization algorithm which uses gyroscope, magnetometer, and accelerometer data to calculate its position, given an initial base location. It features two motors that drive two friction wheels inside the outer shell to propel itself forward and turn in its desired direction. Its design is inspired by the commercially available Sphero [2] while enabling additional data insights and allowing easy addition of sensors.

To accurately design our system, we needed to define our requirements first. The requirements for our system are:

RQ1 The system should be able to navigate to a given location.
- **RQ1.1** It should be possible to set a target through a base station.
- **RQ1.2** The system should roll from its current location to a given point B without any additional assistance.

RQ2 Get real-time feedback from any sensors on a base station.
- **RQ2.1** Wireless communication should be possible from a suitable range.
- **RQ2.2** Sensor data should be sent at speeds of at least 1Hz.

RQ3 Be expandable with different types of sensors

RQ4 Be as small as possible

In this paper, we first describe the background on self-propelling ball design in Section II. Afterwards, we will discuss the design of our system in Section III. Here we will discuss both hardware and software choices. To evaluate our design we conducted a set of experiments which are described in Section IV and their results are presented and discussed in Section V. Recommendations on future work are made in Section VI, and we end with a conclusion of our work in Section VII.

II. Background

A. Self-propelling balls

To move forward in with a spherical robot, torque is needed. The further the centre of gravity is from the centre of the ball, the greater the torque. This means that often the available torque is small compared to the total weight of the ball. An alternative way to propel a ball from the inside is with a rotating mass. Its acceleration will cause the ball to rotate in the other direction, thus generating movement. However, due to the necessity of acceleration here, there is a time limit for the applied torque, so this is not a viable option for moving forward or backward. It can be used for changing direction sideways [3]. A larger diameter of the system increases the potential available torque while decreasing environmental resistance due to small objects[3]. So our goal to make the system as small as possible makes it more difficult to overcome the required torque to drive on different types of terrain. Many designs for ball-shaped robots have been patented over the years[3], and we will cover the designs that provide full steerability. L.R. Clark Jr. et al. designed a steerable ball toy (figure 1). It worked by having a motor (8 in the Figure) rotate an axle connected to both sides of a sphere, and another motor (15) moving a mass to adjust the position of the rolling

axis. Hamster-wheel systems are systems which use traction wheels to apply torque to the ball. These designs typically require higher torque and the driving torque of the ball may be lower than the torque required from the motor driving the traction wheel(s). C.E. Merril et al. (figure 2) and many others (figures 3 and 4) proposed a design which placed a three- or four-wheeled vehicle inside a ball, some using some support inside the ball.

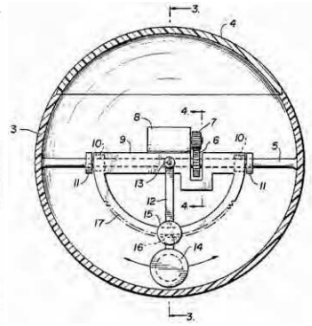


Fig. 1: Steerable robot ball by L. R. Clark Jr. et al.

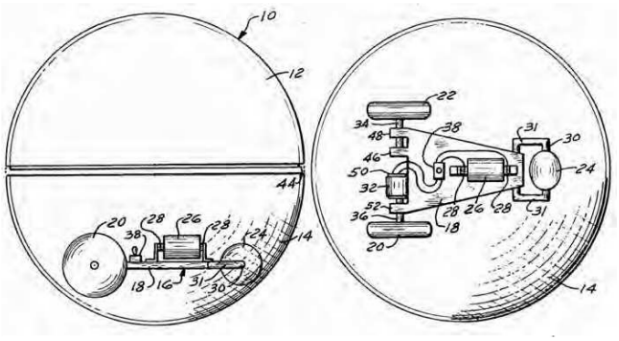


Fig. 2: Steerable robot ball by C. E. Merril et al.

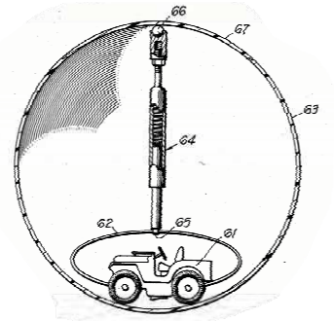


Fig. 3: Steerable robot ball by D. E. Robinson

One- (figure 5) and two-wheeled (figure 6) hamster balls have also been proposed, which rotate a wheel to steer.

The last generation Rollo robot from the Automation Technology Laboratory of Helsinki University of Technology was intended to be used as a home assistant. It included a rolling axis, like Figure 1, but also a rotating rim gear used to change the rolling direction (see Figure 7).

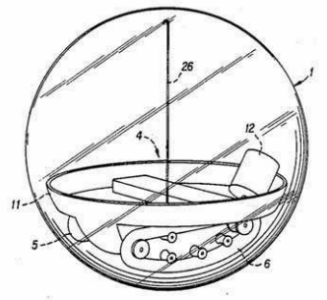


Fig. 4: Steerable robot ball by H. V. Sonessen

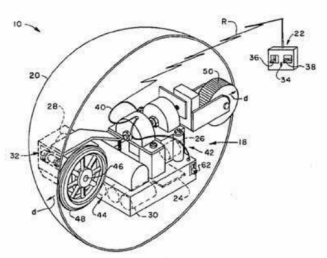


Fig. 5: Steerable robot ball by J. E. Martin

Two more designs for a self-propelling robot ball are by moving weights around inside the ball to move the centre of gravity, or by inflating and deflating cells on the ball's exterior to rotate the ball with instability [3].

The Sphero BOLT[4] and Sphero Mini[2] are commercially available robotic balls. Their mechanical design is similar to that of Robinson (figure 3), with two wheels at the top to balance it inside its outer shell. These bots contain a gyroscope, motor encoders, accelerometers, and LEDs and the Mini is only 4cm in diameter.

III. System design

Our system is designed with a diameter slightly larger than the Sphero Mini, having a diameter of 8cm to meet the requirements outlined in RQ4. The increase in size is due to the fact that we are using off-the-shelf components. These components are carefully selected to optionally fit in a 6cm shell. This leaves room for additional sensors in the 8cm shell and future improvements for the system's size using a 6cm shell.

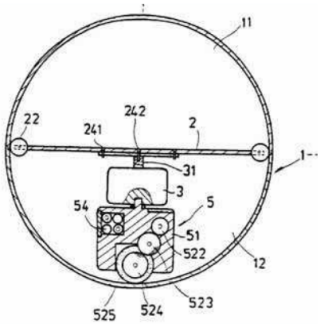


Fig. 6: Steerable robot ball by W-M Ku

Fig. 7: 3rd generation Rollo

A. Hardware Design

To adhere to RQ4, we need the components to be as small as possible. To keep the battery small, we need low energy consumption in our system. Furthermore, for communication, WiFi is more suitable than Bluetooth due to its increased potential range and speeds (given a suitable router is used). So for RQ2.2 and RQ2.2 we need a microcontroller that supports WiFi. This does come, however, with increased power usage as Bluetooth was designed to use less power than WiFi[5]. We think the increased range outweighs its power usage.

1) Microcontroller: The microcontroller should be energy efficient. As our IMU can be communicated with using the I2C protocol, our microcontroller needs to support this protocol as well. The advantage of I2C is that multiple sensors can be added on the same I2C channel, given that the sensors all have unique addresses. This allows for an easily expandable system (RQ3). Furthermore, some sensors use the SPI protocol, so support for this protocol would benefit our system's expandability. Furthermore, it needs to support WiFi communication. In the future, the system may be expanded with a camera, so support for a camera module would be beneficial.

Three suitable options are Raspberry Pi boards, Arduino boards, or ESP boards. Raspberry Pi boards on average have more computational power than their alternatives. However, this of course comes with greater energy usage. Arduino and ESP boards are more lightweight[6]. As our system only has to read a few sensors while controlling two motors, we decided that a lighter-weight board would suit our needs. Non-native camera modules exist which use the I2C interface like the OV7670 CMOS Camera Module [7]. This module is 35x35x30mm, so will be too big for a ball of around 8cm. An alternative is the Petoi Intelligent Camera Module[8], which comes built-in with supported human body, object, and symbol detection and can communicate through serial, I2C, and WiFi. It is much less deep, coming in at 33x33x13mm. However, this product costs 55 euros. The XIAO ESP35S3 Sense[9] is an EPS32 board which comes together with a camera module. The board and camera combination is only 21x17.5x15mm and only costs around 14 euros. The board also supports WiFi when using the included antenna. The antenna is external, but is flexible so can be easily bent to fit anywhere. This combination is thus the most suitable option for our system.

2) Motor system: To move our robot, there are multiple options. First of all, we could use DC motors. DC motors come in two types, brushed and brushless. Brushed motors achieve high torque at low speeds and are about 80% efficient. The N20 geared mini DC motor [10] is a brushed motor and gear package of size 34x12x10mm. There are various gear configurations and axle position options. It can support 3-12V and can spin at up to 1000RPM at 12V. However, brushless motors are more efficient than brushed motors. The lack of brushes also results in more durability[11]. A subvariant of brushless motors is coreless motors. These have a honeycomb structure around a hollow core instead of a more typically found solid core. The advantages of coreless motors are that they have a high power-to-weight ratio, high efficiency and high acceleration and deceleration rates[12]. It is beneficial for our system to have a low centre of gravity. Light motors make this easier, and thus coreless motors seem the best fit. The Bitcraze Crazyflie 2.x motors[13] come in at 7x16mm with a shaft of 3.5mm and can handle up to 4.2V with 14000rpm/V. This would require a gear reduction system but due to its small size, this should easily fit the design. Another advantage is that it is designed to be powered with a 4.2V Li-Po, which also works with the ESP32S3.

Stepper motors are an alternative to DC motors. Stepper motors allow for very precise speed control and have maximum torque at lower speeds. However, stepper motors are typically inefficient as they constantly draw maximum current. A disadvantage of stepper motors is that they require four wires to be operated as opposed to DC motors which only require two[11]. One of the smallest available stepper motors is the M15SP-1N[14], which is 20x15x10.5mm (LxWxH) excluding shaft, which can rotate up to 450 rpm. It operates on 7V. Another small option is the 28BYJ-48[15], which is 35x28x19mm but can't do more than 6rpm at 5V. This is too slow for our use case.

Lastly, there is the option of using continuous servos. Unlike regular positional servos, continuous servos can continuously rotate in two directions based on their control signal. Servos typically have higher torque at higher speeds[11]. Most servos operate at a voltage of 4.8-6V. An advantage of using servos is that it already features gears to reduce the rpm of its motors. Thus we would not necessarily need an additional gearbox. One of the smallest servos available is the DM-S0090D-R[16], which is 23x12x22.5mm. To align their axes, two of these should be placed back to back. This would leave only 15mm for wheels, which would be vertically centred in the ball. For more manoeuvrability, as explained in Section III-B, the wheels should ideally be slightly lower. This is not possible with these servos without the use of an additional gear or belt mechanism, which mitigates their advantage of having included gears.

We decided to use the CrazyFlie motors due to their low weight and small size. Section III-B explains the gear reduction system we used.

As we will be using two motors (as described in Section III-B) we will need a dual h-bridge motor driver. One of the cheapest and smallest motor drivers is the L9110 driver [17]. It is 29.2x23mm, and its connectors and pins can be soldered off resulting in a height of only a few mm. It supports an input voltage of 2.5-12VDC, so no step circuit is required for a 4.2V Li-Po. The Crazyflie motors are rated for 1000mA, but we do not expect to drive them until the stalling. Hence the

driver's rating of 800mA for each channel should suffice. So this motor driver should be a good fit for our design.

3) Localization hardware: To determine the heading of the system there are two main options: a gyroscope or a magnetometer. The attitude and heading reference system (AHRS)[18] requires a gyroscope, magnetometer, and accelerometer to accurately determine the attitude and heading of a system. The accelerometer can also be used to calculate displacement. The MPU9250[19] contains all three of these sensors in a 15x25x3mm form factor and only requires 3.9mA under normal operation.

A more exact alternative for displacement is using a GPS module like the GY-NEO6MV2 GPS module[20]. However GPS modules are not very accurate in indoor usage, and this module can draw up to 67mA of current.

Another alternative is doing localization with the help of additional beacons. These beacons have a known location, and thus by placing a transceiver on the system, the system's location can be determined. The requirement of these additional beacons makes this solution suboptimal.

We decided that the features of the MPU9250 would suffice our use case.

4) Battery: For choosing the right power source for our system we carefully evaluated various trade-offs. In these types of IoT systems, it is very common to select Li-Po batteries due to their performance and size. The choice of the battery is essential for our system, as it directly impacts the device's runtime, weight, and size. The ESP32-S3's power consumption while using WiFi has an average consumption of approximately 100mA. The motors represent the largest power consumption source. Here, we measured a base consumption of 70mA when driving the motors without the gear system and wheels attached. This consumption increased to 250mA when the gear system (see section III-B3 was connected. We expect the complete system to have a higher consumption due to additional load. However, the ball will not be used at full speed, hence we do not expect this maximum estimate in practice. Furthermore, the IMU requires 3.7 mA. We prioritised selecting a battery with sufficient capacity to sustain the system under peak load. However, we also had to consider the physical constraints of the rolling ball design, where both weight and size directly correlate with the battery's capacity. After carefully considering the trade-offs, we settled on a Li-Po battery with a capacity of 650mAh, with a maximum current draw of 500mA, which would suffice our use case, even under peak load. The ESP32-S3 can charge this battery when connected to USB which improves the overall usability of the product.

5) Electrical design: Figure 8 shows how all hardware is connected. The ESP and motor driver are directly powered through the Li-Po battery, whereas the IMU is powered from the 3V3 port of the ESP. Four digital PWM-enabled pins are connected to the motor driver inputs, and the driver's outputs go directly to the two motors. The ESP's SDA and SCL pins are connected to the IMU's SDA and SCL pins, and an external antenna is connected to the ESP as well.

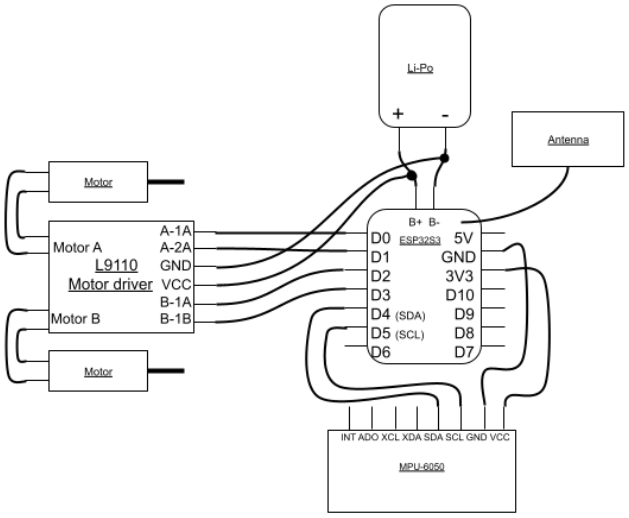


Fig. 8: Electrical circuit schematic

B. Mechanical Design

1) Overall design verification: Looking at the designs in Section II-A, a hamster ball configuration seems most space-efficient due to the absence of a moving mass. Taking inspiration from the Sphero robots, a first prototype was built with two traction wheels and a low centre of gravity and can be seen in Figure 9. This prototype did not have additional support at the top as the low centre of gravity seemed to keep the wheels in the desired position. This prototype was 12cm in diameter as its function was purely to test the propulsion mechanics and identify initial issues. It also contained other hardware than selected in Section III-A. Remote controlling the motors showed this prototype could successfully move forwards, and rotate. It did show the importance of proper balance, as it did not drive in a straight line. It also showed the importance of keeping wires and components off the outer shell, as that caused much unwanted friction which prevented the ball from rolling.

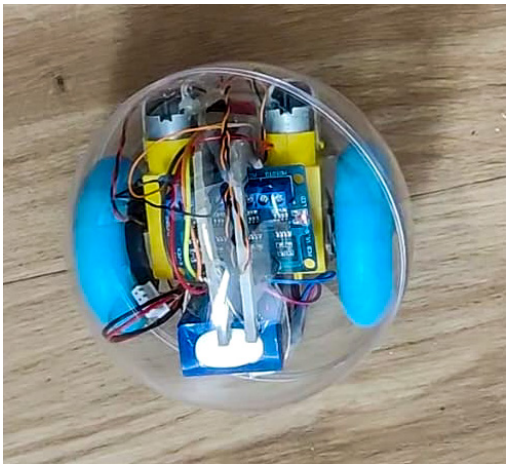


Fig. 9: First hamster ball prototype

We also use two friction wheels in our final prototype, which uses the hardware selected in Section III-A.

2) Friction wheels: The two friction wheels are 22.5mm in diameter, driving on a circular plane of the outer shell of 60mm in diameter. Figure 10 shows a schematic overview of the friction wheel design and the plane they rotate on. These wheels are 3D printed using PLA, and fitted with latex sleeves for grip. We placed them as low as possible as this minimises the amount of torque needed to spin. This means we can lower the voltage applied to the motors, thus decreasing power usage.

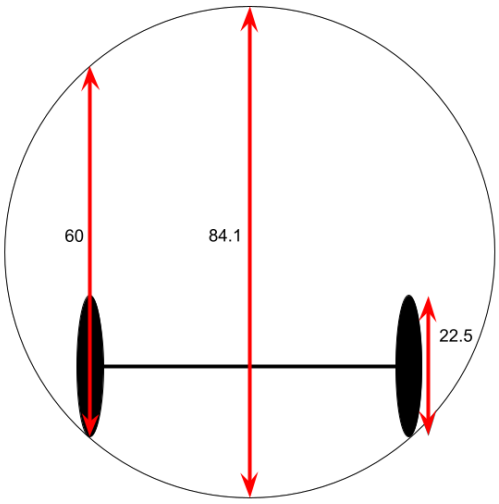


Fig. 10: Schematic overview of the friction wheels and their rotating plane. Measurements in mm.

3) Motor and gear system: As the motors run at 14000 rpm/V, so 58800 rpm at 4.2V, we need a gear system to have the wheels spin at a suitable speed. Each gear system consisted of five plastic commercially available gears. Our design features 2 regular spur gears and 3 compound spur gears. Figure 11 shows an overview of the gears used. The gear ratios are 8:20, 8:22, 10:22, 8:24, resulting in a total gear ratio of 1:44.375.

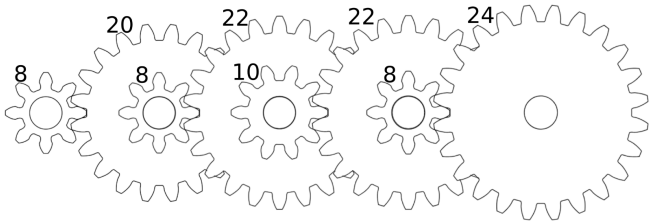


Fig. 11: Gear system used featuring 5 gears. Each gear is labelled with its number of teeth.

4) Hardware fitting: The friction wheels (1 in Figure 12) are rotated by the motors (2) through the gear system (3), located at the bottom half of the outer shell (10). To generate the most potential torque, the centre of gravity should be as low as possible. The heaviest component is the Li-Po battery at 18.13 grams, and should thus be placed as low as possible. The motor and gear systems take up all the space at the bottom, resulting in available space for the battery only slightly below the centre of the outer shell (4). So this will not have much effect on the torque. The motor driver (5) can be found right above the battery, followed by the IMU (6). The IMU's holder has four long vertical slots which are placed over four extrusions to make the IMU as stable as possible. The ESP microcontroller (7) is placed at the front. The bottom is fitted with two weights totalling 17.32g, held by a plastic holder, and two more weights totalling 21.5g (8) taped to the back. This results in a center of gravity down and to the back from the sphere's centre. To keep wires and the WiFi antenna from the outer shell, an inner shell (9) was placed around all components. The two holes at the top of the inner shell are designed to fit additional wheels for balance, just like the Sphero Mini has. however, controllability was proven to be best when no wheels were fitted. The outer shell itself already keeps the ball stable during driving. In the final product, much of the space between the components and the inner shell is filled with wires and the ESP's antenna.

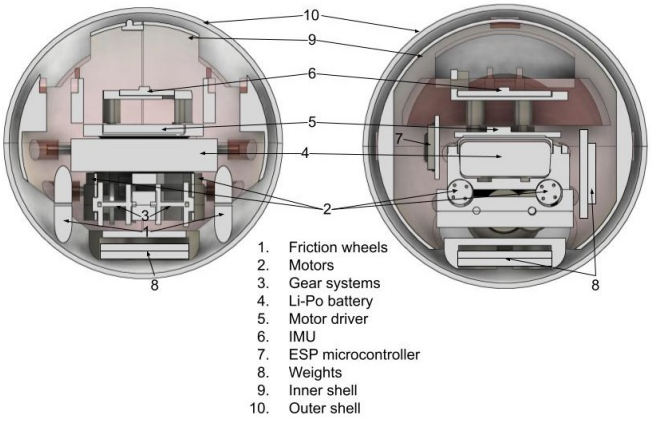
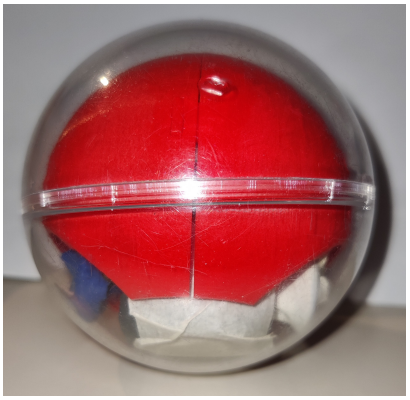


Fig. 12: A cross-section render of the B4W_BB8 system.

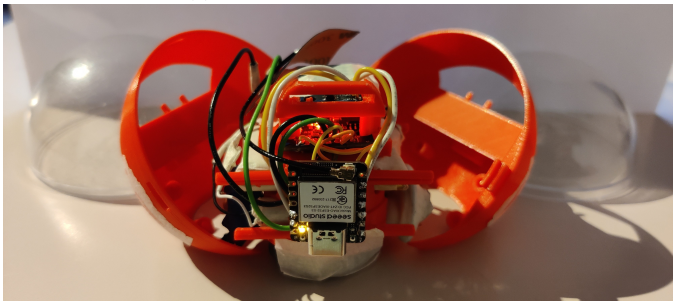
Figure 13 shows a front and opened view of the final prototype of B4W_BB8.

C. Software Design

In this section, we will describe the overall software design and its components. For this project, we agreed on an initial proof-of-concept design and focused mainly on the functional requirements to create a working system. As such, non-functional requirements like security and scalability are not a prime interest and are not considered in this design. First, we will describe the functional requirement that must be included to have a system that performs the requirements stated in Section I. Next, we will describe the selected software frameworks and communication protocols. We will then explain the control algorithms for determining and updating the device's state. The client control panel and the state machine used for communicating information from the device to the client are then described. Finally, the event handling, namely the objective handler and the action handler, are explained in different levels of abstraction. Here, we describe the mechanisms of the ball in terms of high-level objectives, specific action implementations, and the low-level PWM signals to control the individual motors.



(a) A front view of B4W_BB8



(b) Inside view of B4W_BB8

Fig. 13: B4W_BB8

1) Requirements: The device should be able to perform distinctive tasks defined in RQ1 and RQ2 in Section I. The software system is responsible for processing the sensor signals and performing the appropriate actions through the device's actuators. Furthermore, a central control panel should be able to track and control multiple devices. The functional requirements that must be implemented to create the required software system are listed below.

1) The device must be able to execute low-level code and schedule programmable tasks.
2) The device must be able to connect to a network such that it can share its current state and receive new objectives.
3) The device must attempt to reconnect whenever the connection is lost and must do this with exponential backoff to decrease its power consumption.
4) The device and the control panel must be able to use a shared protocol to communicate state updates and new objectives successfully.
5) The device must be able to read the raw sensor data from the connected IMU device.
6) The device must be able to process the raw sensor data and convert this data into practical positional units such as yaw, pitch and roll.
7) The device must be able to process and filter the raw accelerometer data from the IMU device and convert this data to positional changes.
8) The device must include a calibration tool to calibrate the device for accurate measurements.
9) The device must include a unique identifier to enable point-to-point communication between the control panel and the device.
10) The control panel must have input capabilities to capture movement objectives that can be sent to the device.
11) The device must be able to ensure consistency when reading the state vector in each decision round.
12) The device must have an objective handler which processes high-level commands from the control panel and activates specific internal action events to complete the objective.
13) The device must have an action handler which defines each atomic action of the device in concrete actionable steps.
14) The device must be able to express concrete actionable steps as analogue PWM signals to drive the motors.
15) The device must be able to calculate routes from its current location to a target location and process these routes as internal actions using the action handler.

2) Frameworks: The ESP32-S3 microcontroller can be programmed by flashing firmware onto the ESP32-S3 board. This microcontroller supports the popular Arduino framework and ESP-IDF (Espressif IoT Development Framework) [21]. We have selected ESP-IDF as our development framework because it provides low-level access to the microcontroller using the C programming language. It includes many libraries not natively available in the Arduino framework, such as the Motor Control Pulse Width Modulator (MCPWM) peripheral API [22].

ESP-IDF has built-in support for FreeRTOS [23], which is a widely used real-time operating system that has an ample task scheduler. The device should be able to perform different types of tasks, such as message processing, sensor reading, and decision-making. FreeRTOS allows us to focus on the actual implementation of these tasks and relieves us from the low-level scheduling and prioritizing of specific task execution rounds.

3) Communication: The device has communication protocols to support communication between the IMU and the microcontroller and to provide wireless connectivity between the device and the control panel.

The IMU and the microcontroller are connected through a serial communication bus using I2C. We used a publicly available MPU-9250 driver developed for the ESP32 and adapted it to work with the ESP32-S3 [24]. The wireless connectivity between the device and the control panel is established using WiFi. WiFi provides reliable and scalable network capabilities and is easily deployable in our testing configuration. Bluetooth was considered as well since it has an evident advantage in terms of energy consumption [25], but was not selected due to the increased complexity of the code.

The application protocol used to transmit the state and objective data is MQTT [26]. MQTT is an Organization for the Advancement of Structured Information Standards (OASIS) standard messaging protocol for the Internet of Things (IoT). It is a lightweight publish/subscribe messaging protocol that is excellent for connecting remote devices. The device and the control panel both act as MQTT clients and are connected to an MQTT broker located on an external privately owned server. The communication channels are defined through different

topics. A specific register topic is used as the initial communication channel between a device and the control panel. After the device is properly registered to the control panel, the unique identifier of the device is used as the topic to establish a point-to-point connection between the control panel and the device.

4) IMU: The MPU-9250 IMU that is connected to the microcontroller plays a crucial role in the localization and decision-making of the device. The magnetometer is used for determining the heading of the ball. The gyroscope provides data for determining the orientation of the ball in 3D space through roll and pitch angles. The accelerometer is essential for estimating the velocity and total displacement of the ball. The raw data measurements from the IMU must be processed correctly to provide accurate state updates of the device. This happens in different stages, namely the calibration of the IMU, the filtering of the raw data values, and gravity compensation. The displacement estimation requires additional numerical integration methods.

The calibration of the IMU is done in three stages, where each stage calibrates a different sensor. The gyroscope is calibrated by keeping the IMU still to calculate the overall bias. The gyroscope readings can be corrected by subtracting the bias offset. The accelerometer is calibrated by rotating the IMU concerning the X, Y, and Z axis and measuring the gravitational force in both upward and downward directions. Here, when one axis is calibrated against gravity the other axis will be perpendicular to gravity and should have near zero reading. The difference to a perfect zero reading is used as an offset for the respective axis. Lastly, the calibration of the magnetometer is performed by moving the sensor around all axes to find the maximum and minimum values of each axis. The offset and scaling are found by averaging over the min and max values to match the magnetic field of the earth and local magnetic fields.

Inertial Measurement Unit (IMU) filtering is an important aspect of implementing accurate orientation sensors. Many such filters exist to improve the quality of translating raw measurements into useful representations. The Euler angle representation was deemed the most useful for our implementation. AHRS (Attitude and Heading Reference System), an orientation filter proposed by Sebastian Madgwick [27], addresses known challenges associated with Euler angle representations, such as singularities, using a quaternion representation of orientation. The AHRS approach offers several innovative features. It introduces a single adjustable parameter, the beta value, based on observable system characteristics, providing flexibility and adaptability to different scenarios. AHRS also uses an analytically derived and optimized gradient-descent algorithm which provides good performance even at low sampling rates. Furthermore, the AHRS filter includes an online magnetic distortion compensation algorithm. This is important in scenarios where external magnetic fields may interfere with the accuracy of magnetometer readings. The algorithm dynamically adjusts for magnetic distortions to ensure that the orientation estimation remains accurate and reliable. Lastly, the AHRS filter integrates gyroscope bias drift compensation. This is useful because over time gyroscopes may exhibit bias drift leading to inaccuracies in orientation measurements.

The Euler representation, resulting from the AHRS filter, provides heading, pitch, and roll angles. These angles completely describe the device's orientation relative to the earth's magnetic field. The total displacement of the device can be determined from the same raw measurement data. First, we need to get the unit of displacement aligned with the real world. For this, we measured the displacement in centimetres and fitted a model to the perceived displacement. It turns out to follow a non-linear relationship based on a square root function. The model is shown in Figure 14. With the correct unit of the measurements, we can then approximate the velocity of the ball by integrating the accelerometer measurements over time. The same process is repeated by integrating the velocity over time to retrieve the displacement within the given time interval. It is important to note that this approximation is very error-prone, as shown in [28]. One way to make the calculation more reliable is by defining smaller timesteps at the cost of increased computational cost.

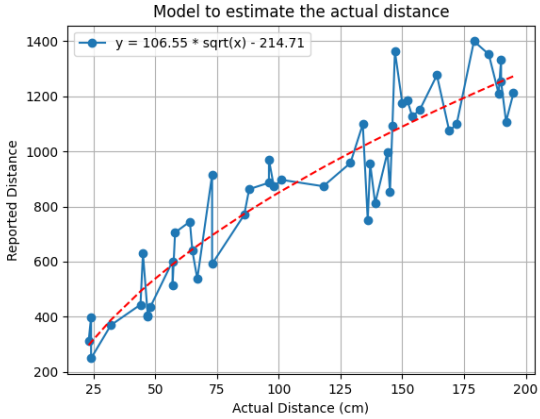


Fig. 14: Model to determine the actual displacement based on the reported displacement of the ball.

5) Event Handling: The software of the ball uses different layers of abstraction in the handling of actionable events. Objectives are events that define large tasks that the ball tries to accomplish. An example of an objective is the task of moving to a specific location. Objectives are broken down internally into smaller sub-tasks called actions. Actions define primitives that can be used to achieve a larger task. Examples of actions are move forward, turn left, or stop. Each action has a specific implementation which drives the PWM signals of the motors.

The interaction between the client and the ball involves a series of coordinated steps to achieve the selected objective. Here's an overview of the workflow:

1) **Control Panel Interaction:** The control panel specifies the objective and defines the maximum duty cycle, providing the initial parameters for the ball's actions.
2) **Objective Handling:** When a new objective is received by the ball, the system processes it through the objective handler, which gracefully concludes any ongoing objective and determines the next course of action to achieve the new objective.

3) **Action Handling:** The determined action is then executed by the action handler, which operates based on predefined logic for primitive actions such as Forward, Backward, TurnLeft, TurnRight, and Stop. Primitive actions directly control the motor driver by generating specific PWM (Pulse Width Modulation) signal patterns, enabling precise motor control.
4) **State Evaluation:** Throughout the process, the system continuously evaluates its internal state, including feedback from sensors like the Inertial Measurement Unit (IMU) to make decisions and send status reports to the control panel.

Several objectives are implemented, including rebooting, driving forward or backward until a stop signal is given, and driving towards a target location. The move-to-target objective is the most complex and important objective that is implemented. It works by calculating the angle and distance to the target position to determine the appropriate action. If the angle is below a specific threshold, the ball moves forward; otherwise, it turns left or right to align with the target. Once close enough to the target, the ball stops and transitions to an idle state, completing the objective.

Each primitive action requires specific implementation details to create the expected behaviour:

- **Forward & Backward:** Gradually increasing or decreasing duty cycles until reaching the target duty cycle. Adjusted duty cycles are transmitted to the motor driver via PWM signals, enabling smooth acceleration or deceleration of both motors.
- **TurnLeft & TurnRight:** Turning involves two distinct phases, including a waiting phase and a pulse phase. The waiting phase pauses the PWM signal to the motors. The pulse phase sends opposite PWM signals to the motors to create a spinning motion of the ball. The waiting phase enables precise control of this action.
- **Stop:** Stepwise reduction of duty cycles until reaching a complete stop. Adjusted duty cycles are sent to the motor driver to gradually slow down the ball motion, with optional measures taken to prevent tilting if the roll exceeds a predefined threshold.

6) Control Panel: The control panel is designed to allow for easy tracking and control of the ball. The panel is created in Python with the TKinter library. It sends objectives to the connected ball and receives periodic status updates over MQTT. When a ball is activated it will register itself to any control panel that is currently active. The control panel consists of an interactive grid, a collection of control buttons, and a status area. Figure 15 shows the design of the control panel.

The interactive grid shows every registered ball on its last communicated location together with its yaw. When a location on the grid is selected, the ball will create a target location on the selected coordinates. The ball can then be tasked to move itself towards the target location using its localisation algorithms.

The control buttons enable easy control of the ball. The buttons send single objectives to the ball. The slider adjusts the maximum duty cycle that is applied to the motors. For

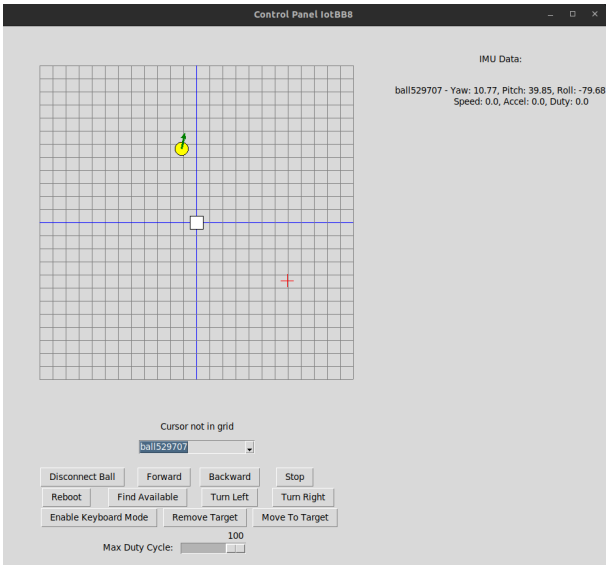


Fig. 15: The design of the control panel.

continuous manual control, the keyboard arrow keys can be used to change the direction of the ball and move the ball forward and backwards.

Every registered ball periodically reports, once every second, its status to the control panel. This information is shown in the status area on the right-hand side of the control panel. The status update is also used as a keep-alive packet by the control panel. When no status update is received for more than 30 seconds, the ball is timed out and removed from the control panel. The ball can re-register when it is available again. The user can also manually search for available balls using the corresponding ”Find Available” button.

IV. Experiment Design

This Section describes the experiments that were used to test the performance of the B4W_BB8 prototype. The experiments cover localization functionality and battery life. All experiments were conducted on plastic flooring.

A. Angle readout verification

To verify if the final prototype can detect the direction it has to navigate to, its angle readouts need to be verified. This was done by placing a printed-out 360-degree protractor around the prototype, rotating it in both directions and verifying the estimated rotation angle with the real-life rotation angle. Figure 16 shows the setup for this experiment.

B. Straight line divergence

The controllability of our system is greatly influenced by its ability to drive in a straight line. This test intends to show its divergence on a straight line. In this test, B4W_BB8 will be positioned at the start of a drawn-out straight line, and its trajectory when moving forward will be monitored.

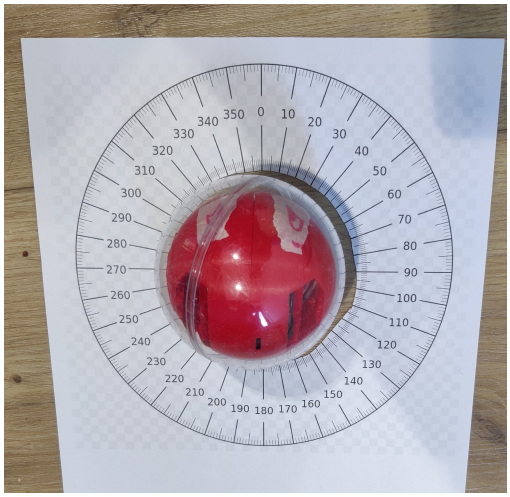


Fig. 16: Set-up for the angle readout verification test

C. Navigation accuracy

To evaluate our system's accuracy in navigating from a given base point to any supplied target, the system was placed in a drawn 3x3m grid. This grid corresponds to the grid in the GUI, which can be overlayed to compare the system's estimated position with its real-life position. We had the ball drive from the grid's centre to four distinct and equally spaced points on the grid (1.5m from each other), and then back to the centre. For each target, we measured the difference between its estimation and the real-life location. Figure 17 shows the test setup and the path the ball is expected to take. The test was conducted two times.

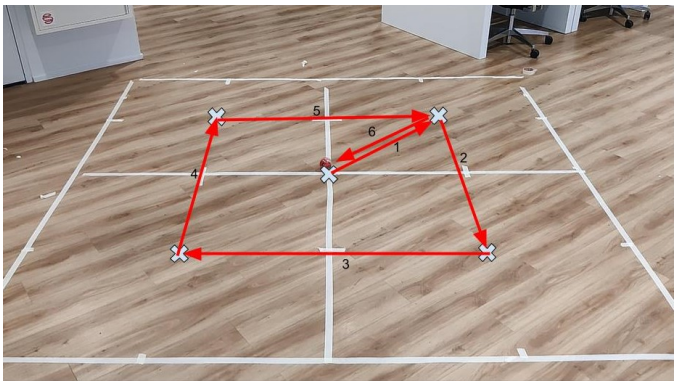


Fig. 17: Set-up for the navigation accuracy test including the expected path

D. Controllability test

To verify if potential navigation errors are due to localization or due to controllability issues, the ball was also manually remotely controlled to take the same path as in Figure 17. However, it was taken in a counter-clockwise direction, so all paths, starting with 6, were taken in reverse.

E. Inclined surface test

For outside applications, our system needs to be able to handle potential changes in elevation in its surroundings. For this reason, its ability to drive up angled surfaces was tested. The ball was positioned in front of a surface with a 5-degree incline, and its performance was monitored while it attempted to drive on the surface.

F. Power consumption

The battery life of the system must be great enough for the system to operate for a significant time. We tested this by running the system under high load (continuously spinning motors) on a fully charged 650mAh battery. The motors were powered with a 46% duty cycle, as that was tested to be the lowest best-performing motor speed. Then we monitored how long it takes for the system to stop working due to the battery running out. To simulate a real rolling scenario, the ball was put on a surface with four ping-pong balls, so it could spin freely (figure 18).

Fig. 18: Set-up for the power test

V. Results and Discussion

A. Angle readout verification

Figure 19 shows the IMU's estimation in angular change compared to the real-life change. We observed that there were differences in the readings, with the largest offset being 25 degrees. It is important to note that errors seem to accumulate over time due to fusing gyroscope and magnetometer readings. Furthermore, magnetic interference from the motors could also cause offsets in the magnetometer's readings.

B. Straight line divergence

Figure 20 shows that our system is unable to drive in a straight line. Two of the three attempts had similar divergence, and the third had a less predictable trajectory. This is mainly due to its centre of gravity not being centred in the ball, which causes it to hang to one side. This offset is caused by not having a specific set place for all components. Connectors are hanging loose and the weights on the back were taped.

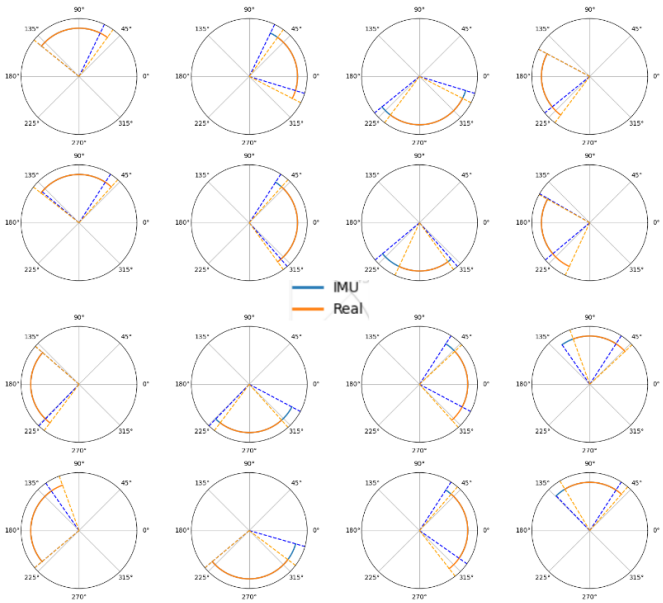


Fig. 19: Results of the angle readout test

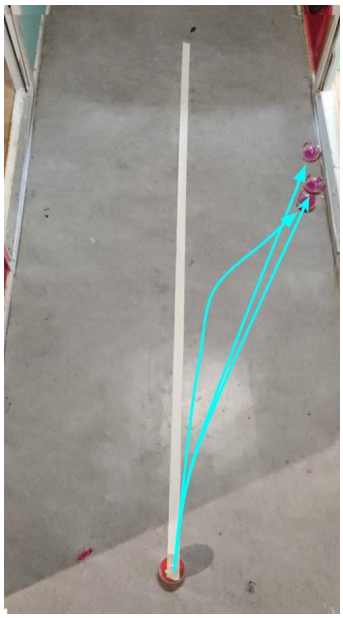


Fig. 20: Results of the straight line divergence test

C. Navigation accuracy

The test was conducted twice. Test session two did not perform the final step of following path 6, and the systems target was set to the centre after path 5 was followed. In both sessions, the estimated locations differed significantly from their real-life locations, and the error became greater over time. Even though our localization algorithm allows a final distance of 20 cm from a set target, the final distance is always greater than this. Tables Ia and Ib contain the estimated versus the real distances from each target. Figure 21a and 21b show the estimated path (red) compared to the real-life (blue). It is unlikely that the differences between estimation and real life in the navigation accuracy tests are due to the straight-line

Path followed	Estimated target distance (cm)	Real target distance (cm)
1	13	50
2	12	67
3	11	128
4	1	88
5	4	127
6	2	181

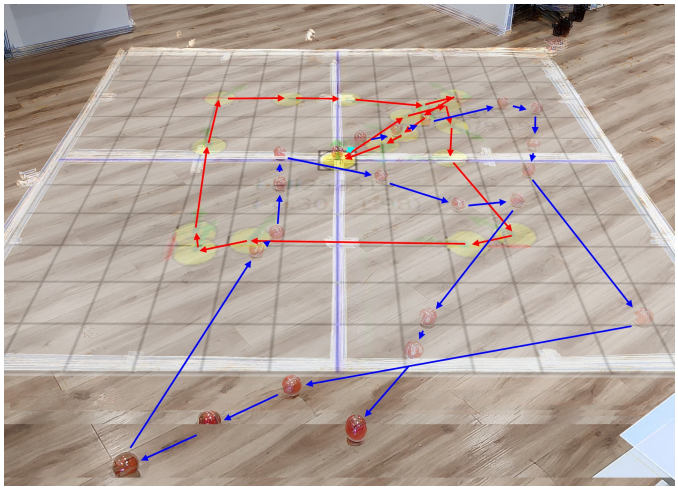
(a) Test 1

Path followed	Estimated target distance (cm)	Real target distance (cm)
1	7	48
2	11	78
3	12	132
4	9	155
5	13	205

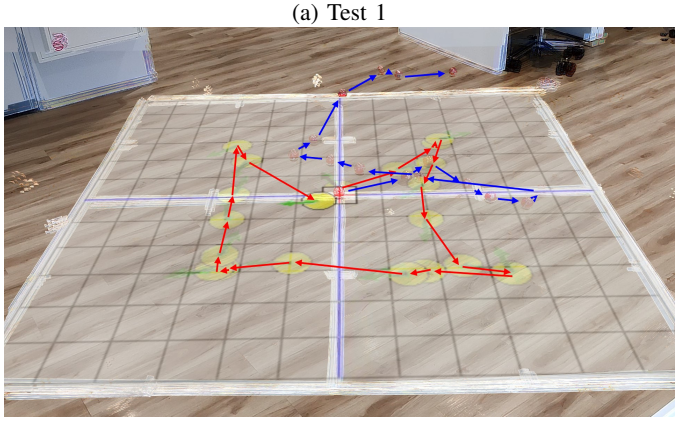
(b) Test 2

TABLE I: Navigation test results. The path numbers correspond to the numbers in Figure 17

divergence as the localization algorithm accounts for this by continuously checking its heading. Angle readouts were also observed to be inaccurate at times. It also appears that our displacement algorithm is not fully accurate and results in a different estimated location from the actual location. Thus over time, the estimated angle between the system and the target will increasingly differ between the real-life angle. This causes the inaccuracies seen in the navigation accuracy test.



(a) Test 1



(b) Test 2

Fig. 21: Navigation test estimated (red) vs real-life (blue) path.

D. Controllability test

We observed that there is some difficulty in controlling the ball remotely. This is mainly due to the command delays over WiFi, which make it difficult to stop in the desired position. Figure 22 shows the path the ball traversed using remote control. It does show that it was possible to steer the ball close to the desired target positions. This test shows that it is possible to manually control the system with reasonable accuracy, even with the end-to-end delay caused by WiFi communication. This promises that even though localization in this set-up is not accurate, controlling the system through an on-board or overhead camera view is possible. This allows the system to still be navigated to suitable positions.

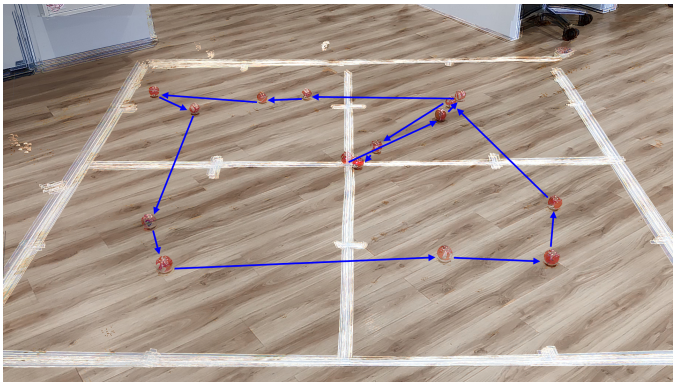


Fig. 22: Result of the controllability test

E. Inclined surface test

It was observed that the system was unable to drive up the inclined surface as it seemed unable to generate enough torque to overcome the resistance. Higher power to the motors only resulted in wheel slip. This result is due to the centre of gravity not being far enough from the system's centre and the friction wheels not having enough grip on the outer shell. Without any run-up, the outer shell would slide over the surface, generating very little traction. This

F. Power consumption

After 1 hour and 29 mins, the motors stopped spinning and the ESP lost internet connection. In a real setting, the motors would only spin for short periods at a time, only while readjusting the ball's position. Further power usage would only be WiFi communication and internal computations by the ESP. Currently, IMU readouts are not paused when idling, but doing so would decrease power usage even more. Be that as it may, the power consumption with a battery of 650mAh is enough for the system to last multiple days with the occasional move, given that only takes a few minutes.

VI. Future work

The evaluation revealed some improvements to be made to our design. Future works should look into alternative ways of measuring displacement. Recommendations here would be looking into ground or flow speed sensors, optical rotary encoders, low-power GPS modules, or localization systems requiring additional beacons. Furthermore, more effort should go into the location of the centre of gravity, by optimizing the location of all hardware, and potentially including heavier weights at the bottom. Additionally, the performance of commercially available motor-gear systems like the N20 geared mini DC motor should be investigated as this can reduce friction within the system, thus decreasing power usage. A custom PCB will improve space efficiency and allow for a smaller system. Finally, an alternative outer shell, which allows the ball to operate on different surfaces as well as inclined surfaces would greatly improve its deployability.

VII. Conclusion

We presented an expandable connected spherical self-propelling system called Balls for Walls: BB8 (B4W_BB8) to allow monitoring in remote areas using mobile systems. Our system can be remotely controlled to navigate target positions with satisfactory accuracy, even though its straight-line accuracy was insufficient. Self-navigation using gyroscope, magnetometer, and accelerometer data proved to be inaccurate in our setup, and errors increased over time. These errors were both due to the displacement calculation and heading estimation, as that was observed to be up to 25 degrees inaccurate. B4W_BB8 is expandable with a variety of sensors, enabling its deployment for many use cases, like intruder detection and environment monitoring.

REFERENCES

[1] S. Sharma, S. Singhal, G. G. S. Kumar, R. V. Prasad, *et al.*, "B4w: A smart wireless intruder detection system," in *ICC 2023-IEEE International Conference on Communications*, pp. 191–197, IEEE, 2023.

[2] Sphero, "Sphero mini - coding robots." https://sphero.com/collections/coding-robots/products/sphero-mini.

[3] T. Ylikorpi and J. Suomela, *Ball-Shaped Robots*. 10 2007.

[4] Sphero, "Sphero bolt - coding robots." https://sphero.com/collections/coding-robots/products/sphero-bolt.

[5] J. Arellano, "Bluetooth vs. wifi for iot: Which is better?." https://www.verytechnology.com/iot-insights/bluetooth-vs-wifi-for-iot-which-is-better\#:\~:text=As\%20a\%20result\%2C\%20Bluetooth\%20is,files\%20like\%20videos\%20and\%20photos., 2023. Accessed: 12-11-2023.

[6] ESPForBeginners, "Differences between arduino and raspberry pi." https://www.espforbeginners.com/guides/differences-between-arduino-raspberry-pi/, 2021. Accessed: 12-11-2023.

[7] Tinytronics, "Ov7670 cmos camera module." https://www.tinytronics.nl/shop/en/sensors/optical/cameras-and-scanners/ov7670-cmos-camera-module.

[8] Petoi, "Intelligent camera module." https://www.petoi.com/products/intelligent-camera-module.

[9] Seeed Studio, "Xiao esp32s3 sense." https://www.seeedstudio.com/XIAO-ESP32S3-Sense-p-5639.html.

[10] Handsontec, *GA12-N20 Datasheet*. https://www.handsontec.com/dataspecs/GA12-N20.pdf.

[11] Helen, "Choosing the right motor for your project: Dc vs. stepper vs. servo motors." https://www.seeedstudio.com/blog/2019/04/01/choosing-the-right-motor-for-your-project-dc-vs-stepper-vs-servo-motors/, 2019. Accessed: [5-11-2023].

[12] Assun Motor, "Coreless motor vs brushless motor." https://assunmotor.com/blog/coreless-motor-vs-brushless-motor/#:~:text=Coreless%20motors%20are%20lightweight%2C%20affordable,for%20automation%20and%20healthcare%20applications, 2022.

[13] Bitcraze, "4 x 7 mm dc motor pack for crazyflie 2." https://store.bitcraze.io/products/4-x-7-mm-dc-motor-pack-for-crazyflie-2?_pos=1&_sid=74c532bc8&_ss=r.

[14] MinebeaMitsumi, "M15sp1n2 stepping motor." https://product.minebeamitsumi.com/en/product/category/rotary/steppingmotor/pm/parts/M15SP1N2.html.

[15] Kiwi Electronics, "Small reduction stepper motor 5vdc - 32 step 1/16 gearing." https://www.kiwi-electronics.com/nl/small-reduction-stepper-motor-5vdc-32-step-1-16-gearing-848?country=NL&utm_term=848&gclid=Cj0KCQjw-pyqBhDmARIsAKd9XIPM20v_T8i6fN9Iq8N-yRV1PEnaDNzpBKOxGTLA0K9RGRUarIFZeasaAmR1EALw_wcB.

[16] BestArduino, "Dm-s0090d-r 360-degree continuous rotation digital servo for robot and uav." https://www.bestarduino.com/p2224/DM-S0090D-R-360-degree-continuous-rotation-digital-servo-for-robot-and-UAV.html.

[17] Laskakit, *L9110 2-Channel Motor Driver Datasheet*. https://www.laskakit.cz/user/related_files/l9110_2_channel_motor_driver.pdf.

[18] W. Geiger, J. Bartholomeyczik, U. Breng, W. Gutmann, M. Hafen, E. Handrich, M. Huber, A. Jackle, U. Kempfer, H. Kopmann, J. Kunz, P. Leinfelder, R. Ohmberger, U. Probst, M. Ruf, G. Spahlinger, A. Rasch, J. Straub-Kalthoff, M. Stroda, K. Stumpf, C. Weber, M. Zimmermann, and S. Zimmermann, "Mems imu for ahrs applications," in *2008 IEEE/ION Position, Location and Navigation Symposium*, pp. 225–231, 2008.

[19] SunFounder, "Imu breakout - mpu-9250." http://wiki.sunfounder.cc/index.php?title=IMU_Breakout_-_MPU-9250, 2020.

[20] Epitran, *NEO-6M GPS Module Datasheet*. https://www.epitran.it/ebayDrive/datasheet/NEO6MV2.pdf.

[21] Espressif Systems, *ESP-IDF Programming Guide*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/.

[22] Espressif Systems, *ESP32-S3 Peripherals API Reference: Motor Control Pulse Width Modulator (MCPWM)*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/api-reference/peripherals/mcpwm.html.

[23] FreeRTOS, *FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions*. https://www.freertos.org/index.html.

[24] Psiphi75, *GitHub Repository: esp-mpu9250*. https://github.com/psiphi75/esp-mpu9250.

[25] G. D. Putra, A. R. Pratama, A. Lazovik, and M. Aiello, "Comparison of energy consumption in wi-fi and bluetooth communication in a smart building," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–6, 2017.

[26] MQTT, *MQTT*. https://mqtt.org/.

[27] S. Madgwick, R. Vaidyanathan, and A. Harrison, "An efficient orientation filter for inertial measurement units (imus) and magnetic angular rate and gravity (marg) sensor arrays," tech. rep., Department of Mechanical Engineering, April 2010.

[28] A. Blake, G. Winstanley, and W. Wilkinson, "Deriving displacement from 3-axis accelerometers," in *Computer Games, Multimedia Allied Technology*, 2009. Computer Games, Multimedia amp; Allied Technology ; Conference date: 01-01-2009.