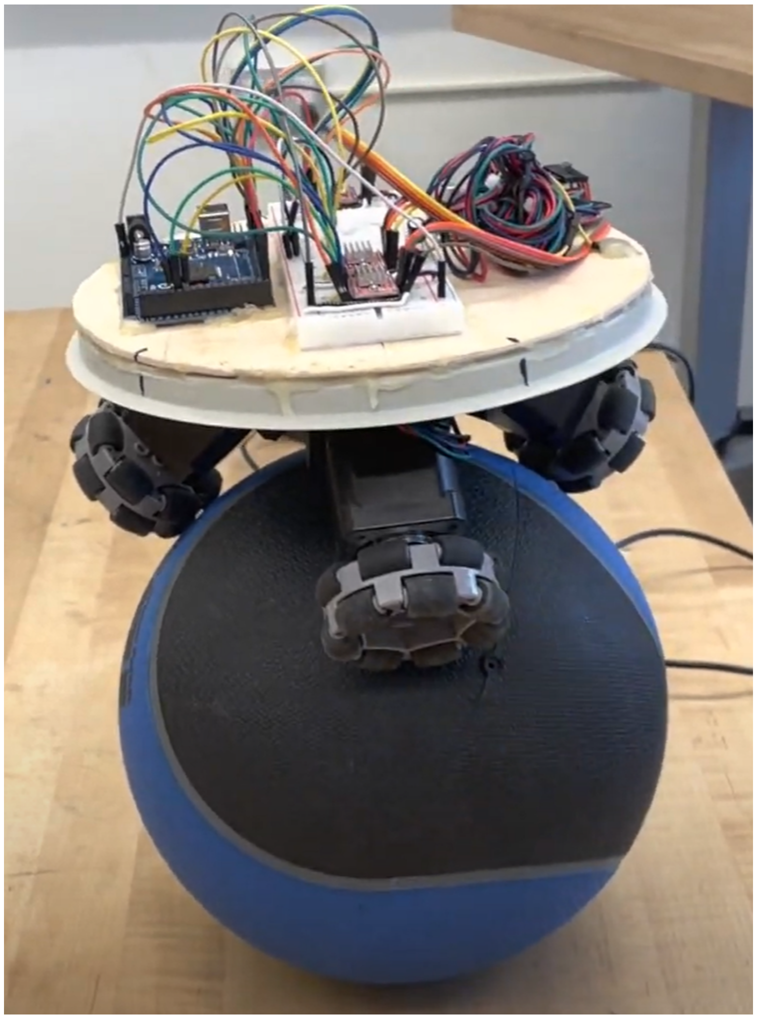
The Ball

Ismail Ouazzani Chahdi
Student# 1006804095
ismail.ouazzani@mail.utoronto.ca

Alex Alexiev
Student# 1006971252
alex.alexiev@mail.utoronto.ca

Luis Vazquez
Student# 1007211330
luis.vazquez@utoronto.ca

Christopher Adolphe
Student# 1007073113
christopher.adolphe@mail.utoronto.ca



1 Introduction

1.1 Project Overview

The project we have chosen to carry out consists of an electro-mechanical mechanism that is placed on the top of a ball which regulates itself to remain balanced on the top of the ball despite disturbances. This robot resting on the ball consists of three motors positioned in a polyhedron manner in which the actuators control the maneuvering of the robot with respect to the ball. While the ball is in motion, to ensure the mechanism remains balanced on top, the actuators will be activated to counteract the ball's movement. An Arduino Mega is used to calculate the necessary control responses bawsed on the feedback from the main sensor, namely the Inertial Measuring Unit (IMU). The actuators will be instructed to move accordingly by the Arduino board.

1.2 Project Goals

- Our personal goal for this project is to apply the theoretical knowledge we have gained in this course. It is important for us to solidify our understanding of certain subjects discussed in class by applying it in practice to a real-world opportunity.
- The project goal is to have our robotic system balance on a ball on its own and attenuate disturbances from the environment.

1.3 Applications of Project

Applications:

- Point-to-point: One possible application is to make the entire ball-robot system follow and translate from point A to point B. This is done by tilting the robot in the direction it wants to go and stabilizing it at that angle. Consequently, the ball-robot system will move in the desired direction.
- Movable Table: By adding a flat surface to the top of the robot body, the entire system can become a balanced table that moves around as desired by the user.
- Carrying Robot: Another possible application of this project is to add a flat surface to the top of the robot, for example a tabletop, and place objects on it. One can then apply a force to this tabletop which rests on the mechanism to introduce a motion to displace the object. The robot in this situation will counteract the disturbance and self-balance on the ball which causes the ball-robot system to 'follow' the motion desired by the human.

2 Background

2.1 Changes since Project Proposal

- Planned Changes to Improve the System

  Initially, the ball of choice for the project was a bowling ball, given its size, weight, circular shape and lack of features to disrupt the traction of the wheels with the ball (eg. grooves). However, after doing further research it was deemed that heavier objects have a tendency to over-damp the system, being stable on its own due to high inertia. As such, the bowling ball was switched out for a lighter basketball. As we wanted to retain the perfect circular shape that the bowling ball had, we selected a basketball with the shallowest grooves. Furthermore, the leather material on the outside of the basketball, compared to the polyester material on the bowling ball would allow for a higher coefficient of friction between the wheels and the leather on the basketball. This would in turn reduce potential slipping not only between the wheel and the ball but also the ground and ball, thereby resulting in smoother movements as observed from the human eye.

  Our original proposal had planned to use a Raspberry Pi to calculate the control responses from the input sensor. However, after deliberation with our advisors, it was deemed that this project could be realized using other means, particularly solely with an Arduino. From research, it was also determined that implementing application-specific fast controls loop is better handled using simple platforms.

- Changes Driven by Issues with Equipment and Facilities (eg. equipment, facilities)

  At the time of the proposal, we had planned to print the motor couples using a 3D Printer, which was successful and implemented. However, we had also planned for the top plate, upon which the Arduino and breadboard with the IMU, would be attached to its top side, and the motor couples would be bolted to its bottom side, would be laser cut. However, given the high demand of the laser cutter, the plate was unable to be printed in time of project delivery. As a result, this structural component was intended to be replaced with a flat circular plate that could be drilled in to allow for the bolt holes. In the end, our group was able to find an object meeting this description given we ended up using the lid of a plastic paper bin.

Finally, we wanted to ensure that the final controller would not just be compatible for one type of ball, but be robust enough to deal with balls of different sizes, materials and weights. As such, throughout the testing process, various balls such as basketballs, volleyballs and light medicine balls were used to ensure the final design was strong enough to deal with many types of balls. These balls were also tested on different terrains to add further rigour to our testing process.

2.2 Theory

In this project, the design challenge of balancing the mechanism on the ball was broken down into two inverted pendulum control problems in orthogonal directions, the first in the horizontal, left and right direction, and the second in the vertical, back and forth direction. The tilt angles of the device in the x and y directions was measured by the gyroscope situated on the breadboard on the top of the robot. These angles will serve as the control input to the system. Using the angles of the robot about the x and y axis, velocities for the mechanism in the x and y direction will be calculated to regulate the system back to its desired balanced, equilibrium position of resting at the very top of the ball. This x and y velocity in the frame of the robot will be converted into individual wheel speeds using simple geometry.

To achieve the desired attributes to properly implement the two inverted pendulums in the x and y directions, numerous course concepts have been implemented. Mainly timers, interrupts, registers, control loops, and PID.

Interrupts were used in two main aspects of this project. First, we used the hardware timers built into the Arduino Mega to trigger interrupts at a 10kHz frequency. On each ISR for this timer we performed simple logic to check if the stepper motor should be stepped or not depending on the desired speed we wanted to realize given current error displacement angles from the equilibrium position. Secondly, we configured the MPU6050 IMU to generate an interrupt whenever it had new gyro data available. We configured it to report data at 100Hz.

Being able to modify registers to configure chips was a crucial part of this project. Namely, the MPU6050 IMU we are using has a built in DMP (digital motion processor) that is capable of fusing the raw linear acceleration and angular velocity data into a quaternion that completely represents the orientation of the device in the world frame. However, most libraries that interface with this device don't take advantage of this feature and leave the motion processing up to the main processor (the Arudino Mega in our case). Although the library we ended up using handled the configuration of the device it was helpful to understand register assignments to ensure we were taking full advantage of the MPU6050 gyro.

In this project, another course concept that played a major role was the use of PID controllers. Control loops were used to calculate the control output, namely the speed of the three motors connected to the omni wheels which were in contact with the ball. The IMU mounted to the top of the robot was able to calculate the angle with which the mechanism was tilted with respect to the x and y axis. Using these two angles, a PID controller could be implemented to both these angles to regulate the mechanism to always remain on the top of the ball and thus have both these angles with a desired angle of 0 degrees. The error in this PID control would thus be the angle by which the mechanism was tilted in the x and y direction. Once applying an appropriate PID control in both directions, three corresponding outputs to balance the mechanism would be sent as a signal to each of the three stepper motors.

Initially, a P control was applied to try to stabilize the system. As this controller was tested on numerous balls, it became clear that this proportional P constant depended on the weight and size of the ball. With this P control, we observed that the mechanism had very unsmooth and harsh movements, especially around the equilibrium point. As such, we attempted to filter out the effect of the small angles by only having proportional control if the displacement angle in either the x or y direction was greater than a particular threshold. Thus if the angle was, for example, less than 2 degrees with respect to the x and y axis, no control output would be produced. This technique was attempted for numerous thresholds, however, it was found that this was not a very successful idea given typically when the mechanism surpassed roughly 8 degrees in the x or y direction, it would fail due to the gravitational force of the structural components of the mechanism which counteracted the stabilizing force of the control of the motor wheels. As such, this dead zone would cause the mechanism to often react too late.

Optimization was then attempted by implementing the derivative and integral term. We anticipated that the integral control could remove constant position errors, while the derivative term could speed up the response by decreasing settling time. Implementing a complete PID control would result in an output equation of: (Page 51 of lecture 11).

$$Y[i] = K_p E[i] + K_i \int E + K_d \frac{d}{dt}(E) \tag{1}$$

where the error signal, E, represents the angle measured by the IMU by which the robot is displaced with respect to the equilibrium point in the x and y axis. Furthermore, as we would need the derivative for the current time, we could only apply backward divided difference. Given the real time speed needed for these control inputs, opting for increased accuracy and stability at the cost of longer computation time would not make sense to have a higher order derivative. However, despite testing numerous sets of different PI, PD and PID controls with various gains, we discovered that the P controller produced the best results. Upon reflection, we believe that the derivative term did a poor job due to noise in the sensor measurements the IMU took. When there is large changes in error in the system, the derivative of this error may amplify the signal by a large amount. Taking the average of two or three IMUs mounted on the mechanism would have helped with this issue. Furthermore, we believe that the integral term was ineffective due to windup as exemplified by the increased overshooting.

As a result, we ended up using a P controller for the final implementation. We were able to perform all the angle calculations on the IMU which allowed for increased performance, which will be discussed in greater detail in the Final Design section below.

3 CONTROL DESIGN

3.1 CONTROL EQUATIONS

ω_x, ω_y, ω_z represent the angular velocity of the ball with respect to an axis, as shown in figure 1. We can relate the angular velocities to the horizontal velocities of the ball via:

$$\begin{aligned} v_x &= \omega_x R \\ v_y &= \omega_y R \end{aligned} \tag{2}$$

Let v_{s1}, v_{s2}, v_{s3} represent the the velocity input of every motor. We did research and obtained the input for every motor with respect to the horizontal and angular velocities using the equations in [1]:

$$\begin{aligned} v_{s1} &= \quad -v_y \cos\phi + K_z\omega_z \\ v_{s2} &= \left\{+(\sqrt{3}/2)v_x + (1/2)v_y\right\}\cos\phi + K_z\omega_z \\ v_{s3} &= \left\{-(\sqrt{3}/2)v_x + (1/2)v_y\right\}\cos\phi + K_z\omega_z \end{aligned} \tag{3}$$

where $K_z = -r\sin\phi$ is the coefficient of rotating the ball in a fixed position, about the z rotation axis, and ϕ is the angle between the vertical and the wheels positioned on the ball. Our main goal is to balance the robot on the ball, so we can set ω_z to zero.

The corrections required for v_x and v_y are proportional to the angles θ_X and θ_Y measured by the IMU. Therefore, we implemented a P controller:

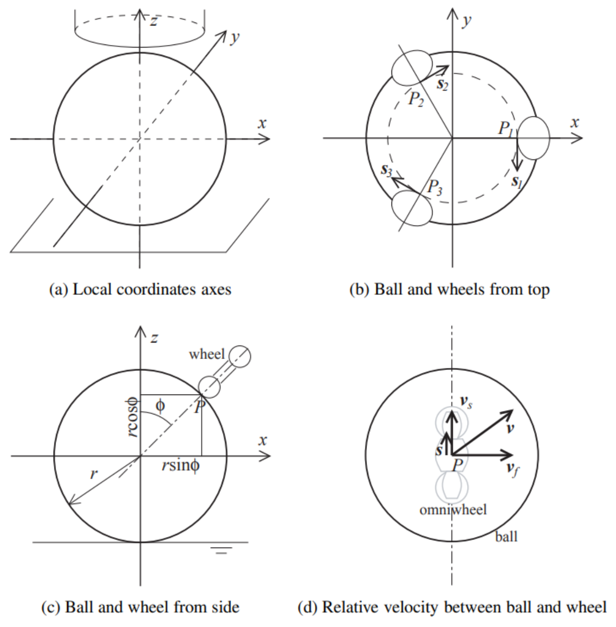
$$\begin{aligned} v_x &= K_p\theta_x \\ v_y &= K_p\theta_y \end{aligned} \tag{4}$$

We found that the best gain for our hardware was $K_p = 0.3$.

4 FINAL DESIGN

4.1 MICROCONTROLLER

To choose our microcontroller our main concern was the number of pinouts and having built-in hardware for I2C communication. This was because each stepper motor controller required 5 digital pins (1 step, 1 direction, and 3 micro-stepping). The Arduino Mega has 54 digital pins which is more than enough and leaves room if we want to add



z
y
x
(a) Local coordinates axes
y
x
P_1
P_2
P_3
s_1
s_2
s_3
(b) Ball and wheels from top
z
wheel
ϕ
P
rcosϕ
rsinϕ
r
x
(c) Ball and wheel from side
v_s
v
s
v_f
P
omniwheel
ball
(d) Relative velocity between ball and wheel

Figure 1: Axis definition and ball-wheel relationships, from [1]

more sensors later. The Arduino Mega also has SDA and SCL pins, which allow for the use use of I2C sensors like the MPU6050.

Another benefit of the Arduino Mega is that it has 6 built in hardware timers, two 8 bit timers and four 16 bit timers [2]. These timers allow us to use interrupts to call a function at even intervals which can be very useful so that critical tasks (like driving the stepper motor) don't get blocked. The 16MHz clock on the Mega provides enough resolution for over 10kHz interrupt frequency.

Some other advantages of the Mega are that it can accept 7-12V input which allows us to power it directly from the power supply that powers the stepper motors. Furthermore, using a Mega results in access to the Arduino community which has a lot of resources and forums that can help with problems we might encounter. As well, there are countless libraries available for common tasks like reading data from the IMU which simplifies the code we have to write by providing simple interfaces for us to use.

4.1.1 SENSORS

The only sensor we needed was an IMU. We do not need encoders on the motors because stepper motors enable precise position (and thus speed) control with no external hardware given we control when it steps and know the size of the steps. When looking for an IMU, we selected the MPU6050 because it is cost effective, and also one of the only IMUs available that has digital motion processing built in. It has 16 bit ADC for both gyro and acceleration data, and can output angle data at a maximum of 200 Hz which is more than required as past attempts at this project used less than 100 Hz for their sensor data. This IMU communicates over I2C and is widely used so it has a lot of support and libraries that are easily accessible. When testing the sensor we observed that it took a 5-10 seconds for the position data to stabilize and then it did not drift at all after 2 minutes of stationary and shaking motions. Furthermore, we observed the precision of the sensor to be less than 0.01 degrees which would not be a significant source of error compared to other factors in our device like surface grip and surface irregularities.

4.1.2 ACTUATORS

To choose our actuators we had to decide between DC and Stepper motors. DC motors might appear simpler to control because one can simply change the input voltage to adjust the speed. However, to achieve precise velocity control, which was necessary for this project, we would need to add an encoder to the motor and tune another PID loop for velocity control. The stepper motor provides the benefit of not needing a position encoder or a PID controller for velocity. Instead, as you can control each step and are aware that each step is 1.8 degrees, one can purely apply open loop control and achieve really high accuracy. Stepper motors are also very easy to mount because most of them have a rectangular form factor. Stepper motors also have comparatively high torque which will be useful due to the heavy weight of our entire robot mechanism. We settled on the Nema 17 stepper motor [3] because it has a holding torque of 3.2 kg-cm at 1.2 amps. The next size-up stepper motor would be too large that mounting it would present challenges on a small object like a basketball.

To control the stepper motor we went with the A4988 stepper motor driver [4]. We chose this controller due to team members' familiarity with using it in a past course, namely ESC204. Furthermore, the ability for it to perform micro stepping and split the 1.8 degrees per step into sixteenths was crucial because it allowed for smoother control over the speed of the wheels without jittery motions. In addition, it has a max current of 2A which is greater than the motor's stall torque current of 1.2A. The low cost of this motor drive was also appealing given we were able to attain five A4988 with carrier boards for only 12 dollars on Amazon.

4.2 Software

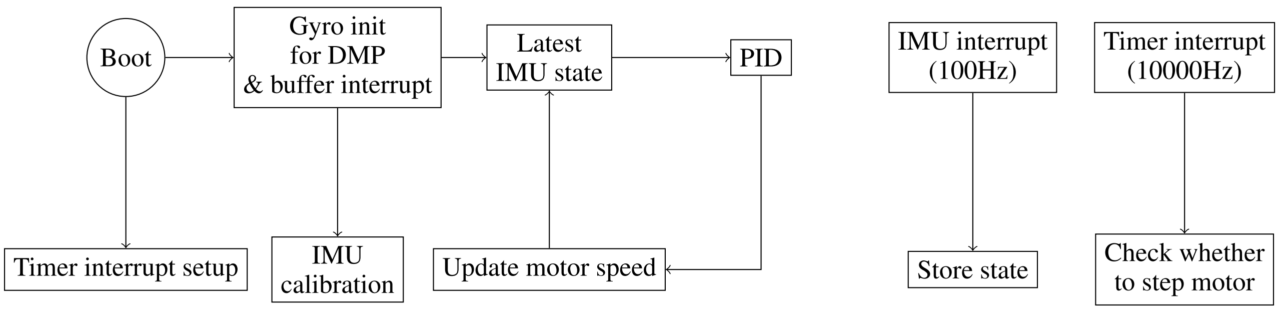


Figure 2: Block diagram of the system.

4.2.1 Level of Language

We decided to use C++ because it allowed us to create classes to improve the readability and usability of our code. Also, there were many libraries such as the hardware timer interrupt and IMU libraries which were supported in C++. We did not need to do lower-level programming like assembly because we could achieve the desired loop rate without too much code optimization, but rather from optimization of the high-level structure of the code.

4.2.2 Design Patterns

The code structure was split into three parts: Velocity control of the motors, reading IMU data, and a PID control loop.

Motor control:

To precisely control the velocity of the motors we simply need to change the time between each step the stepper motor takes. The stepper motor steps on a positive or negative edge of the digital direction pin. Therefore, if we have a large delay between steps, the motor moves slower, and if there is a small delay between steps, the motor moves faster. Therefore we have the relationship for the stepper motor to be

$$v \propto \frac{1}{delay}$$

Knowing this means for a given velocity, we can calculate what the delay between steps should be. Once the delay is calculated, we use a hardware timer interrupt running at 10kHz to decide when to step. To do this, each time in the ISR we check if enough time (the desired delay) has passed since the last step, and, if so, we perform a step and reset our counter. We used a library [5] to take advantage of the onboard timer interrupts on the Arduino Mega. This library let us set the frequency and then we just provide it an ISR. It took a lot of testing to arrive at this solution. Initially, we were only using a delay in our main loop, however quickly realized this would not work given other code in the main loop takes time to execute and this execution time would change the time between steps, thereby nullifying the entire argument for the precision of the stepper motors. There were a few parameters we had to adjust with this code, namely the update frequency for the timer interrupt, the calculation for the delay, and the type of micro-stepping we did (half step, quarter step, etc.). We experimented with all these parameters and made some key observations. First, if we made smaller steps then the motor would run smoother, however, we would need to run the timer interrupts at a higher frequency because you need to take more steps per second. In the end we settled on eighth steps because we observed smooth motor motion and the other code in the project wasn't impacted by the timer interrupt. Once we settled on eighth steps, we estimated a max speed the motor would need to turn and calculate the delay needed for that speed. With our first goal being to balance the robot on the ball and not move translationally, we would not need to move the motors very fast because it should only be making small adjustments. In the end, we chose 250 microseconds as the delay for our max speed (this would be the smallest delay). Then we just calculated the delay as:

$$delay = \frac{250}{v}$$

Where v is a number between 0 and 1. We decided not to use SI units at first because we just wanted to make the ball balance so contrived units would suffice. Later, if we wanted to follow trajectories we would need to start using SI units. However, this would not be a large change because we know the angle change per step and the wheel circumference.

IMU raw data:

The raw data from the MPU6050 (or any gyroscope) is the angular velocity and linear acceleration for each axis. This alone is not very useful because we need the absolute angular orientation of the robot to know when it is level, otherwise, we would only know its velocity. Therefore some motion processing has to be done which mostly involves integrating the rotational velocity to get the rotational position. The linear accelerations are also used to reduce the error in this integration because it can measure the acceleration of gravity downwards which acts as a known reference. The MPU6050 has built in digital motion processing which does exactly this and is capable of outputting the rotation in degrees about each axis, which is what we want. Once configured to perform this, it does the calculations at the preset frequency and then stores them on a FIFO buffer of size 1024. When a new data point is added it triggers an interrupt in our code which stores the x and y angles into global variables. We were able to adjust the digital motion processing rate which would effect how often we received values. We found that 100Hz was fast enough for the PID controller to balance on the ball. However, the MPU6050 can go up to 200Hz if we need faster gyro data in the future. The IMU communicated with the Mega over the built in SDA and SCL pins for I2C. We used a library [6] for the configuration of the IMU for onboard DMP, and using I2C to read from the FIFO buffer. All the specs for the IMU are from its datasheet [7].

PID Loop:

Finally, we have a PID loop running in the main loop of our program. It uses the latest gyro data to calculate individual velocities for the motors using the formulas outlined above for control. Reading the gyro data simply involves accessing a global variable, and then setting the motor speed involves changing a variable in the motor class, both of which are very fast operations. The exact frequency of the PID loop was not too important because we were mainly limited by the 100Hz of the IMU.

4.2.3 Possible Modifications

The biggest room for improvement we have is in the motor speed control code as there were a lot of things we wanted to test but did not have the time to perform. First, we were curious if we could simply use the built-in PWM pins in the Arduino Mega to control the motor stepping by setting a 50% duty cycle and adjusting the frequency based on the desired velocity. This would greatly reduce the load on the main program because the Arduino has PWM optimized already where as we are essentially creating our own PWM generator that is probably not very optimal and ends up blocking the program. Another option would be to have an entirely separate microcontroller just to control the motors due to the high precision they need for their stepping frequency.

4.3 Structure

The goal of the structural design of this project was to manufacture an easy-to-use and customizable robotic system capable of properly balancing on a ball, on which it will sit. There were three main structural components as part of the design, namely the wheels, actuator and plate.

Wheels: A big consideration for the design was the number of wheels and motors we would need to counteract the disturbances. We settled on three actuators to help the simplicity of the system and stay within budgetary constraints. It is important to note that using three wheels resulted in the fact that one wheel could be perpendicular to the direction of motion and prevent the correction to occur. This is reason to support the design choice to use omni wheels. These wheels allow the rotation of the ball in all directions since in the case of a wheel being orthogonal to the direction of motion, the smaller wheels on the omni wheel will roll accordingly.

Actuator Positioning: The point at which the three wheels touch the ball occurs at exactly two-thirds of the ball's height. We found that this height lets the actuators have enough control over the rotation of the ball and still allows for the removal of the robot from the ball with ease. In addition, the location of the wheels ensures that the omni wheels are tangential to the surface of the ball.

Plate: The plate was designed with the primary focus of having the accelerometer and gyroscope sitting exactly in the center of it in order to get the most accurate measurements. Moreover, the design can be expanded upon by adding more levels to the plate by simply using standoffs between plates. This accommodates for future applications as previously discussed.

A 3-D model of the entire structure of the robotic mechanism can be observed below.

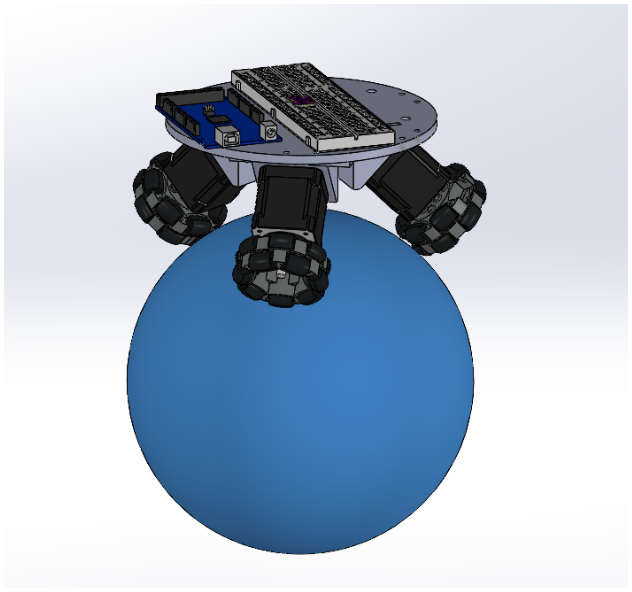


Figure 3: A CAD model of our final design

4.4 User Interaction

With the goal of making the robotic system easy-to-use, the user has to perform few steps to get the robotic mechanism ready for performance. Firstly, the user has to coonect the robot to a power source via either a portable battery or a power outlet on the wall. Secondly, with the robot, and thus IMU, newly powered, the IMU, as mentioned above will need time to stabilize. This stabilization time was observed to be roughly 5 seconds. As such, once powered on, leave the robot resting on a flat surface oriented upward, as it would when resting on the ball. Finally, place the robot mechanism on the ball with all four wheels touching the ball and the robot centred on the top of the ball, as much as possible. At this point, the robot is now ready to perform its operations.

4.5 Analysis

4.5.1 Unused Features

- Integral and Derivative Controller As mentioned above, despite iterations on the type of controller implemented, the P controller was found to be the most effective. However, despite being more complex, PI, PD or PID controllers often are more effective than only P controllers. As such, if there was more time, it would be beneficial to look into mitigating noise for the derivative term and look into strategies to reduce the effects of integral windup. These should help to improve stability of the system by reducing rise time, overshoot and steady state error after disturbances. Visually, these should make the mechanism smoother.
- Angular Velocity around Z Axis Capability Despite incorporating a term for the mechanism to spin around the z axis with respect to the ball, no tests were conducted to verify this feature. As such, given more time

our team would like to perform testing and iterate on this feature to work alongside the current balancing capability of the robot.

4.5.2 Future Iterations

Furthermore, if provided more time for this project, there are many possible modifications we can think of to either improve the system or give it another purpose. Firstly, we plan on replacing the plastic top plate with a quarter-inch acrylic plate to make the system more sturdy and noise resistant. Secondly, the use of multiple accelerometers and gyroscopes and averaging both their measurements could help tell the actuators how to respond with a lot more accuracy and overall enhance the performance. Lastly, we would like to make this system wireless by adding a LiPO battery such that the robot does not need to be connected to the wall plug.

In regards to possible modifications to the purpose of the system, we could implement certain improvements such as a camera to carry out computer vision-requiring tasks. Another idea would be to create another system to combine them both and treat it as a robotic bike.

- Increase Number of IMUs Used:
  One of the issues that was observed during testing was the noise in sensor readings from the IMU. This resulted in poor results from adding in the derivative term in the PD and PID controllers tested. If two, or even three, IMUs were mounted on to the top of mechanism, as opposed to only one, sensor fusion could be used (like a Kalman filter) to provide more accurate results, which would be less impacted by noise. As a direct result, the use of the derivative term would be improved. Please note, the original plan was to use three IMUs, however, delivery delays resulted in the IMUs not being available by the time of the project report deadline.
- Make the System Lighter
  Whenever the mechanism reached an angle in the x or y axis that was greater than roughly seven degrees from resting directly on the top of the ball, the system was often seen to be unable to recover. The weight of the system was a major reason for this as the gravitational force always will be acting in the opposite direction of stability. As such, with more time, it would be beneficial to investigate ways in which to make the entire system weigh less or ways in which to reduce this destabilizing force.

5 Performance

5.1 Tests to Validate the Design

Some of the testing procedures used to validate the final design include:

- Checking for stability when there are no disturbances in the system
- Checking for stability when an external direct force (hand push) is applied to the top plate
- Checking for stability when the surface upon which the ball rests is slanted
  All of these tests, were performed using different types of balls, namely volleyball, basketball and light medicine ball, as well as on different surfaces, including a wooden table, a cement floor and grass, to further validate the integrity of the final design.

5.2 Achievement of Design Goals

Overall, the design worked as intended. Most notably, the CAD showed a good representation of the actual ball-robot system which ultimately allowed us to ensure the robot had a good grip on the ball to control its motion. The location of the accelerometer and gyroscope was exactly in the center using our CAD design for reference which helped get the required measurements and minimize error.

5.3 Software Iterations

Motor code:

As discussed above we were initially using delays in the main loop to control the stepper motors, but this was impacted too much by other things in the loop so the frequency and thus speed was jittering and variable. Upon implementing the timer interrupt for motor stepping at precise intervals we were able to achieve much smoother motion. We also noticed a weird effect where for very small velocities below 0.001, the delay would become really large and the effect is that the motor would exhibit strange effects like oscillations that we weren't able to diagnose the root cause of. As a result, we limit the minimum motor velocity to 0.01 to prevent this and ensure the motor turns smoothly across its entire velocity range.

IMU:

Initially, we were only reading raw values from the IMU without using its onboard digital motion processing. Then we were using the Madgwick attitude and heading reference system library (https://github.com/arduino-libraries/MadgwickAHRS) to calculate the orientation on the Mega processor. This was simpler initially because we could easily adjust to the parameters of the algorithm if it wasn't performing properly. However, when we were running the motors at a very high update frequency they would start blocking the arduino. Motion processing requires a fixed and known timestep between data measurements but this was varying due to the motor control (once again why we want to try using PWM for motor control in the future). We were trying to avoid doing the motion processing on the IMU initially because a lot of forums said the documentation was quite poor for which registers we need to set up and the firmware we need to load on the IMU. After a lot of digging, we found an easy-to-use library that takes advantage of onboard motion processing. Once we implemented this the motion processing would no longer be affected by any other parts of our mechanism so it was much more robust.

After slowly figuring out these two main issues over the course of the project we were able to get the robot to balance on the ball. It was a very slow and long process because we had to keep re-tuning the PID to see if it was able to balance with the current frequencies of the sensors and motors and we kept changing those values. Once we checked that each part of the project (motors, IMU) was operating very smoothly and robust, we dedicated a while just to tune the PID and were able to make the robot balance.

5.4 Hardware iterations

The most significant hardware issue we faced was the positioning of the IMU given that its readings depended on it. In the early stages, we noticed a lot of jiggering and sometimes moving in a specific direction as the robot attempted to balance. When we hard-coded an offset for the measurements we found that the performance improved significantly. This lead us to realize that although the IMU was mounted on the breadboard, its soldered pins were at a slight angle from the pcb introducing a physical offset. To fix this we bent the pins slightly to create a right angle in relation to the IMU's PCB. However, when the robot was powered the jiggering was still present. To further enhance the robot's performance we fixed a plywood piece to fill the gap between the IMU and the breadboard and taped it down. This removed all possible displacements of the IMU due to the system's motion and overall proved to be very effective.

5.5 Constraints Due To Environment

There were no significant constraints for the system due to the environment, aside from the fact that the ball needed to be placed on solid ground (ie. not placed in water) and connected to a power source.

6 Conclusion

In this project, we were able to apply our theoretical knowledge of microcontrollers, communication protocols, sensors, and actuators and put it into practice. This experience allowed us to solidify our knowledge and learn new things by approaching a difficult task. In the end, we were able to get the robotic system to balance by itself on a ball with minor disturbances using an accelerometer and gyroscope as our sensor and three stepper motors as our actuators. In the future, we plan on implementing another IMU to make better measurements and improve the behavior of our system.

References

[1] M. Kumagai and T. Ochiai, "Development of a robot balanced on a ball–first report, implementation of the robot and basic control–," *Journal of Robotics and Mechatronics*, vol. 22, no. 3, pp. 348–355, 2010.

[2] M. T. Inc., "8-bit avr microcontroller with 64/128/256k bytes of in-system programmable flash." Microchip Technology Inc., 2012.

[3] Components101, "Nema17 stepper motor." Components101, n.d.

[4] Pololu, "A4988 dmos microstepping driver with translator." Pololu, 2011.

[5] K. Hoang, "Timerinterrupt." GitHub, 2021. https://github.com/khoih-prog/TimerInterrupt.

[6] ZHomeSlice, "Simple_mpu6050." GitHub, 2018. https://github.com/ZHomeSlice/Simple_MPU6050.

[7] I. InvenSense, "Mpu-6000/mpu-6050 register map and descriptions." InvenSense, Inc., 2012.