IJRCS

International Journal of Robotics and Control Systems
Vol. XX, No. XX, 2022, pp. XX-YY
ISSN 2775-2658
http://pubs2.ascee.org/index.php/ijrcs

ASCEE

Development of a 3D Printed Biologically Inspired Monoped Self-Balancing Robot

Denis Manolescu[a1], Emanuele Lindo Secco[a2*]

[a]Robotics Laboratory, School of Mathematics, Computer Science and Engineering, Liverpool Hope University, Hope Park, L16 9JD, Liverpool, UK
[1]20203547@hope.ac.uk; [2]seccoe@hope.ac.uk
* Corresponding Author

ARTICLE INFO

Article history
Received Month xx, 20xx
Revised Month xx, 20xx
Accepted Month xx, 20xx

Keywords
Biomimetic robotics;
Biologically inspired robotics;
Legged robots;

ABSTRACT (10PT)

The development of lightweight, agile robots built with a high degree of efficiency and precision is a complex and particularly challenging task. In Robotics, the perspectives of such systems have the potential to reimagine and reevaluate space and planetary explorations while enhancing the human ability to reach and analyze different environments. This research evaluates ways to build a single-leg robot capable of self-balancing on any terrain with the prospect of omnidirectional saltation steering. This report will go through the process of designing a 1-degree of freedom, balanced robot while evaluating various hardware options in motion control and data feedback. Based on these research findings, this paper will also discuss the difficulties and issues encountered and the approach to solving them.

This is an open-access article under the CC–BY-SA license.

1. Introduction

NASA initiated the research on jumping robotic systems in the late 1960s onwards with the scope of gathering data and finding new methods of efficient locomotion on the lunar surface. The Lunar Pogo Stick [1] and the Lunar Hopping Transporter [2] were just a couple of interesting concepts introducing the notion of hopping through the lunar regolith by taking advantage of the low gravity on the Moon. Throughout time, engineers took inspiration and tried to emulate the jumping mechanism of animals, where natural evolution successfully created a balance between the necessity to survive and the need to conserve energy for later use (Fig. 1 and Fig. 2). The locomotion system of these animals formed by bones, muscles and tendons can be modelled as an open loop spring-mass-dumper system [3].

The versatility of such systems offers a variety of dynamic movements, depending on the type of gait desired (Fig. 1). In terms of surface contact, the gait represents the sequence in which the system in motion or the footfall touches and lifts from the ground up [4].

From the perspective of a robotic system, the jumping locomotion has to be performed by an energy-efficient structure with a very short-time energy density [5]. Compared with other types of mobilities, saltation movement allows for adaptability to an unpredictable environment, a solid ability to overcome rough terrain and obstacles, fast steering, high point-of-observation and long latency for decision-making events. The jump control consists of modelling the robot around essential parameters like angle of attack, take-off direction, velocity and stability, righting reflex, landing buffering and self-correction mechanisms [6].

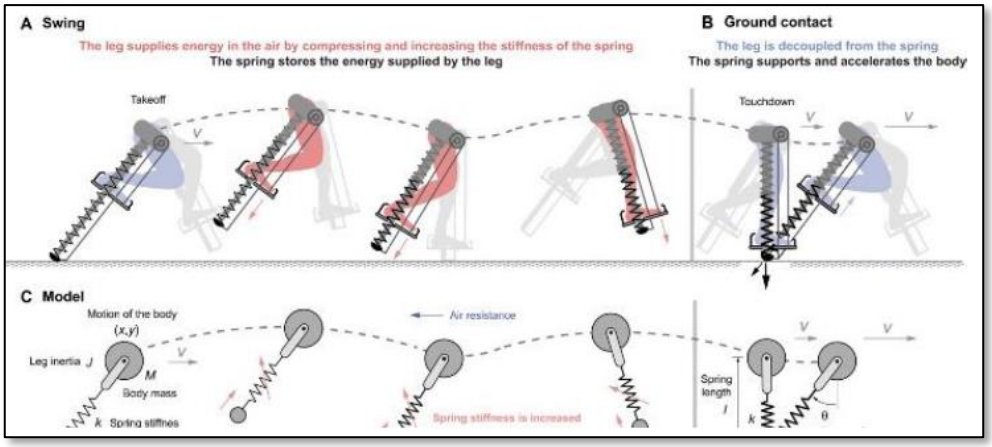

Fig. 1. The human-leg motion modelled, i.e. a spring-mass-dumper model, according to A Sutrisno, 2020 [2].

As a reference, at the writing time of this paper, there are a few robotics developments that have successfully reached a high level of complexity and functionality and pushed the study of jumping locomotion a bit farther: Spot from Boston Dynamics [7], Salto 1P from UC Berkeley [8], Leonardo from Caltech [9], Ascento 2 from ETH Zurich [10] and Hugh-Flying Jumper from University of California [11] (Fig. 3).

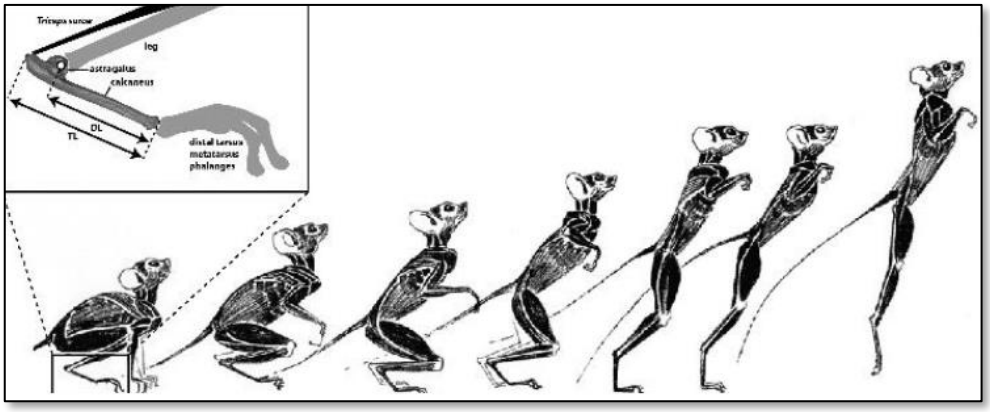

Fig. 2. The Galago Senegalensis aka Bushbaby motion diagram from DM Boyer, 2013 [12].

This paper will cover the first stage in the development of the jumping robot, exploring different design and motion mechanisms, sensors, controlling techniques with data gathering and interpretation and how it can achieve a balanced position. This work will also take inspiration from the previous design of other robotics arms developed in the lab [13, 14]. In particular, we aim at designing and prototyping a monoped robot whose design is biologically inspired from current literature and effective natural design. The project will follow these steps:

- Selection of the components
- Hardware design
- Software design
- Integration
- Preliminary testing

Accordingly, the research contribution of this paper are (i) a novel design of monoped robot with biologically inspired feature and (ii) the integration of Hardware and Software solution for the development of self-balancing capability.

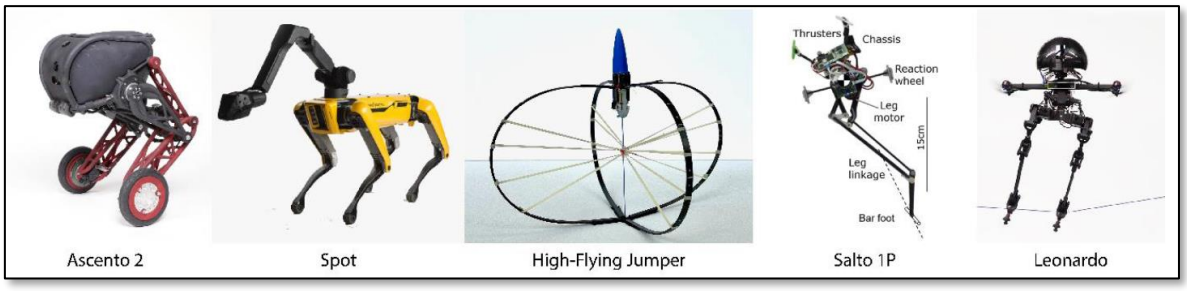

Fig. 3. Some of current Robots with jumping capabilities (September 2022): from the left hand side to the right, the Ascento 2 (ETH), Spot (Boston Dynamics), the High-Flying Jumper (University of California), Salto 1P (UC Berkley) and Leonardo (Caltech).

2. Materials & Methods

The next section of this paper, about the general aspects and Design, will talk about the 3D design process of the robot and the motivation behind its present form. The development process presentation will continue with the components or Hardware paragraph, discussing individual components, their performance within the system and their viability. The programming & algorithm section (i.e., the Software) covers the data processing, the control algorithm method, and other significant programming aspects of the robot.

The findings of this research, including encountered issues and solutions, will be presented in the Results paragraph. The same section will also discuss technical decisions for the future development phase. In the final segment, the Conclusion will reflect on the development process and future work and ideas.

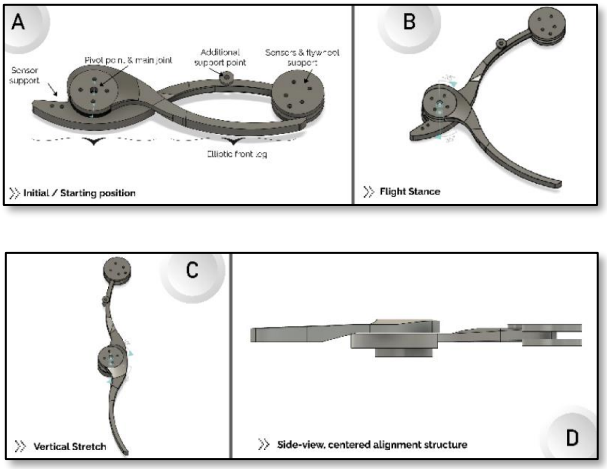

Fig. 4. The 3D model of the proposed Monoped Robot: a two-element compass-like system is made of a knee-joint plus two rigid parts (i.e., the femoris and the tibias) in a biologically inspired fashion.

3. Design

The 3D design of the project is built using Autodesk Fusion 360. The latest version of the robot has a full vertical stretching length of 34 cm (Fig. 4, panel C) and an initial horizontal starting position length of 23 cm (Fig. 4, panel A).

The structure of the robot consists of two parts:

The elliptic front leg and the elliptic support hull – the latter part can also bear support for most of the main components, including sensors, microcontroller, reaction wheel, and encoders. The weight of the 3D printed parts is 40g, and the whole weight of the robot is 190 g, including all components (Fig. 5, panel D).

The 3D design is made to center the entire structure of the robot linearly; this aspect brings benefits in terms of stability, modelling, and calculus (Fig. 4, panel D). The elliptical shape concept of the robot structure has been carefully analyzed. It represents an abundant data point that seems to favor efficiency in propulsion and landing by exploiting inertia and gravity.

4. Hardware

At this stage, the central processing unit and the brushless DC motor controllers are not integrated into the robot body because of weight concerns. The next version of the robot will address this issue.

The Arduino Mega 2560 microcontroller

The central computing power of the robot is generated by an *Arduino Mega Rev3* board built around the *8-bit Atmega 2560* microcontroller produced by Microchip. The board operates on 5 V and has a clock speed of 16 MhHz while offering a convenient 54 digital connection pins and 16 analog pins (Fig. 6, panel A).

In this development stage, having a higher number of pins helped speed up the testing and tuning of the major components. Having two separate I2C pinouts has undoubtedly made a difference.

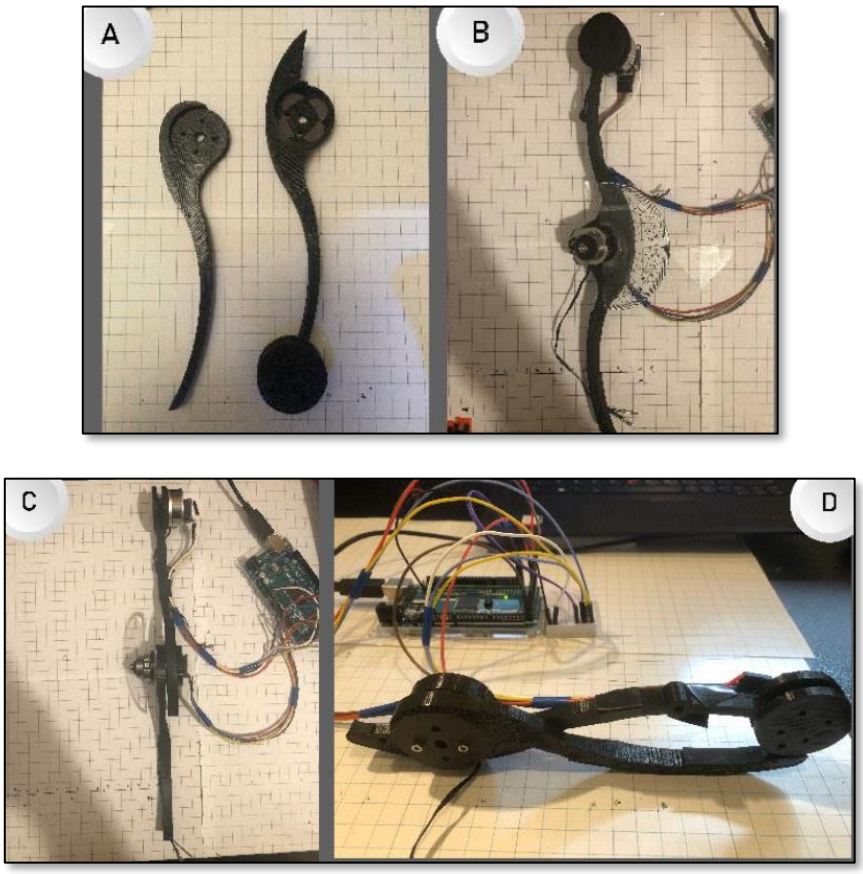

Fig. 5. The 3D printed components combined with some of the Hardware components.

A Simple FOC BLDC Arduino Shield

The brushless DC motor controller shield used in building the robot is the *SimpleFOC V2.0.4* from *Makerbase* (Fig. 6, panel B). This controller is specially designed to be attached on top of each other, on top of the Arduino. It works with an external 12-35 V power supply while offering extension access to the I2C via SCL and SDA pins and encoder/hall sensor interface [15].

As mentioned before, the weight of the robot needs to be kept under strict control; for that reason, both the shields and the Arduino are connected to the system from a distance.

The shield is a cost-effective and fast option to control the precision of brushless DC motors when it works in tandem with an encoder. This device is a community-based open-source project that offers well-documented C++ libraries to gain access to all the board features.

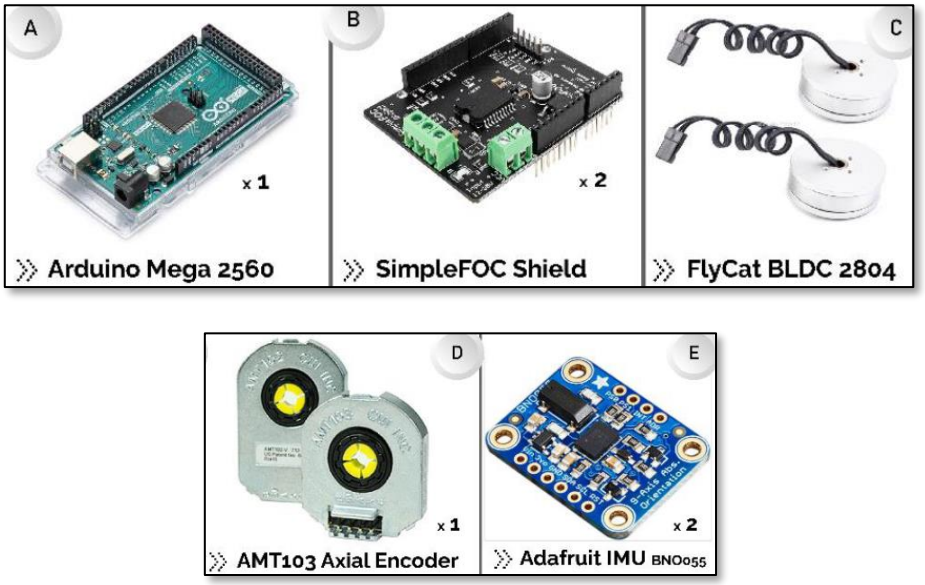

Fig. 6. All the Hardware and Electrical components for the robot manufacturing and integration.

FlyCat 2804 Brushless Gimbal Motors

The joint of the robot is built using the *Flycat 2804 Gimbal BLDC Motor* (Fig. 6, panel C). This 42 g actuator has seven pole pairs and offers 140 kV, which is erkele 1680 rpm and a maximum torque of 0.23 N/m. It is set to operate on 11-14.8 V.

The same motor is set to be tested and used for the reaction wheel, but this aspect will be covered in future research.

The position of this actuator is critical because it determines the angle of attack and the flywheel rotational behavior while in stance mode, flight mode and landing mode. The precision of the joint motor is first kept in check by the *AMT103 Rotary Encoder*, then re-checked by the *Adafruit IMU*.

Additional tests have been performed by a second BNO055 attached on the same axis, on the elliptic support hull, more on that in the Algorithm and Results sections below.

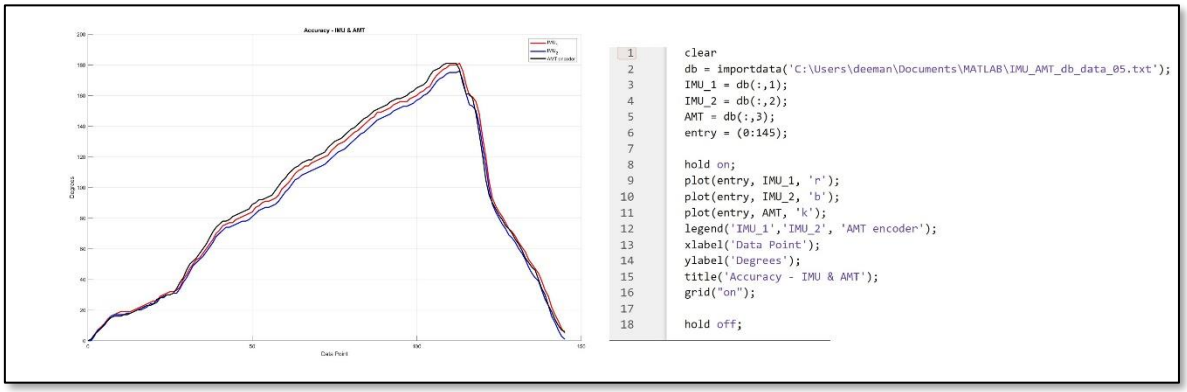

Fig. 7. Plotting of the IMU and encoder data with the Matlab Programming Language (The MathWorks Inc®) – 2 x IMU sensors & 1x AMT223 encoder.

AMT103-V Incremental Axial Encoder

The joint of the robot integrates on top of the actuator shaft an *AMT103-V Rotary Incremental Encoder* provided by *CUI Device* (Fig. 8, panels B and C).

This axial encoder weighs 41g, operates at 5 V on three channels, and connects to the *FOC Control Shield*. The encoder provides the first line of data feedback about the position of the motor. The second line of data feedback is given by a 9-DoF absolute orientation IMU attached to the back of the actuator (Fig. 8).

Access to the data stream is provided via the FOC shield in correlation with the C++ libraries developed by the AMT manufacturer.

The test results of this technique will be explained in depth in the next paragraph.

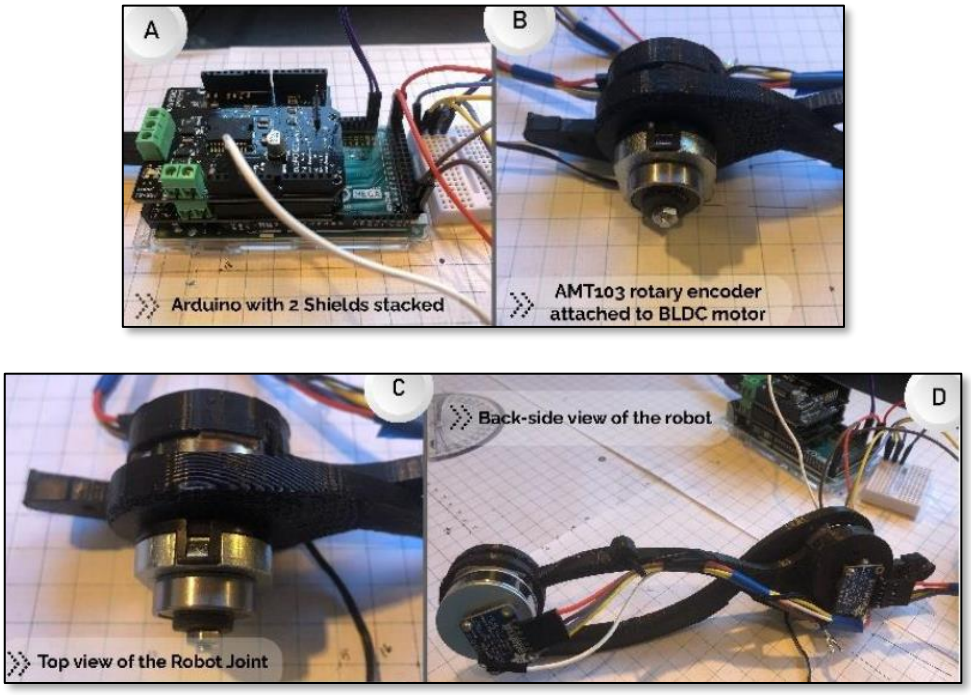

Fig. 8. The main control unit (panel A), the joint (panels B and C) and the rear view of the robot (panel D).

Adafruit 9-DoF Absolute Orientation IMU BNO055

The *Adafruit BNO055 Inertia Measuring Unit* (IMU) is an intelligent 9-DoF sensor fusion developed by Bosch. The three grams sensor integrates a full range of algorithms with an *Arm Cortex-M0* data processing chip (Fig. 4, panel B). It operates on a 3.3 V and outputs data via I2C protocol.

Another important aspect is the capability to program and change the IMU IDs and the memory data registers, making it easier to work with many BNO055 sensors simultaneously.

The A*dafruit IMU board* offer full coding community support with a decent number of C++ libraries and examples available via GitHub or directly integrated into Arduino IDE support [16].

In the testing stage, the level of precision recorded between the two identical BNO055 sensors, as seen in the layout shown in Fig. 7, panel D, has been outstanding, reaching 96-97 % accuracy. Most of this difference in data received was caused by the vibrations or displacement from not being properly screwed into the robot frame.

Quark_01_jumping_robot_project_draft

```
#include <SimpleFOC.h>  // BLDC motor controller library

// BLDC & driver instance. Formats: BLDCMotor(pole pairs). BLDCDriver3PWM(pwmA, pwmB, pwmC
BLDCMotor motor = BLDCMotor(7);
BLDCDriver3PWM driver = BLDCDriver3PWM(9, 5, 6, 8);

// Global variables
float targetPos;          // variable to drive BLDC in radian
float defaultPos = 1.5;   // default BLDC movement

// Instantiate Commander interface
Commander command = Commander(Serial);

// Set & get functions for target commands
void doCCW_target(char* cmd) { command.scalar(&targetPos, cmd); }
void doCCW_default(char* cmd) { targetPos = defaultPos, command.scalar(&targetPos, cmd); }
void doReturnHome(char* cmd) { targetPos = 0, command.scalar(&targetPos, cmd); }

// Main configuration function
void setup() {
  driver.voltage_power_supply = 12; // driver config. power supply voltage
  /* Note: Limit the driver max dc voltage to protect low-resistance motors */
  driver.voltage_limit = 6;
  driver.init();

  // Link the motor & the driver
  motor.linkDriver(&driver);

  // Trial & error setup: Start low, well under 1Amp
  motor.voltage_limit = 3;    // [V]
  // Set the transition velocity between target angles
  motor.velocity_limit = 0.6; // [rad/s] cca 50rpm

  // Open loop control configuration
  motor.controller = MotionControlType::angle_openloop;

  // Initiate motor functions
  motor.init();

  // Setting target commands
  command.add('d', doCCW_target, "CCW target angle");
  command.add('w', doCCW_default, "CCW 1.5 Radians");
  command.add('s', doReturnHome, "return to start position");
```

Fig. 9. Connection setup between Arduino – Simple Foc – Motor

5. Software

The entire project has been programmed within the Arduino IDE environment in C++. Here we have two main blocks of code (Fig. 9 and Fig. 10):

- The first block of code is designed to coordinate the *IMU* data feedback by setting the connections with and returning the data to the *Arduino Mega*.
- The second block of code uses the *SimpleFoc* shield programming features to set up the connection between the motors and the Arduino and capture the *AMT103 Encoder* data for the actuator position feedback loop.

The user can interact through the serial terminal by typing a command. Here there is the option *(i)* to drive the motor to a default position given in radians and allocated by the *defaultPos* variable, or, alternatively, *(ii)* to drive the actuator to the desired angle assigned by the *targetPos* variable. Both options can continue with a *doReturnHome()* function to rotate the motor to its initial position.

To help get control of the motor, the developer offers a simple C++ library <*SimpleFOC.h*>, easily scalable and updated by the open-source community (*line 1*). Next is to set the instances for the motor (*line 4*) and the *FOC Controller* (*line 5*) for memory allocation. An essential aspect in this stage is to understand well the motor requirements and the manufacturer specifications, that is because the motor object function *BLDCMotor()* needs as an argument the number of magnetic pole pairs. In this case, the motor is built with 14 magnets, resulting in 7 pole pairs.

The driver object function *BLDCDriver3PWM* takes three connection pin numbers as arguments, the 3-channels that generate the pulse-width modulation signals, and the enable pin number.

```
54 // Main loop
55 void loop() {
56   // The open loop angle movement function
57   motor.move(targetPos);
58
59   // User communication
60   command.run();
```

Fig. 10. The Motor control main loop.

```
monopo_dual_IMU_sense_good2go §
1  #include <Wire.h>
2  #include <Adafruit_Sensor.h>
3  #include <Adafruit_BNO055.h>
4  #include <utility/imumaths.h>
5
6  /* Set the delay between fresh samples */
7  #define BNO055_SAMPLERATE_DELAY_MS (100)
8
9  /* Define sensors (id, address) */
10 Adafruit_BNO055 bno_1 = Adafruit_BNO055(55, 0x29);
11 Adafruit_BNO055 bno_2 = Adafruit_BNO055(56, 0x28);
```

Fig. 11. The Adafruit IMU BNO055 libraries set-up.

Code line 12 defines the instance for the *Commander interface*. This basic GUI is a monitoring, configuration and control interface designed by the FOC developers to sustain their hardware in the Arduino environment. This interaction feature uses the user input as predefined commands that can be typed in the Arduino serial monitor window. The functions from code *lines 15-17* define the three options the user can operate the motor with:

- move to the default position

- move to the desired position or
- return to the home position.

Next, in the Setup section, *voltage_power_supply* variable holds the value of the FOC shield operating voltage, 12V. And, through trial and error, the motor *voltage_limit* and *velocity_limit* variables get set (*code line 30-32*).

```
/* Displays some basic information on this sensor */
void displaySensor_1(void)
{
  sensor_t sensor;
  bno_1.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor #1:       "); Serial.println(sensor.name);;
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution);

  /* Get the system status values (mostly for debugging purposes) */
  uint8_t system_status, self_test_results, system_error;
  system_status = self_test_results = system_error = 0;
  bno_1.getSystemStatus(&system_status, &self_test_results, &system_error);

  /* Display the results in the Serial Monitor */
  Serial.println("");
  Serial.print("System Status: 0x");
  Serial.println(system_status, HEX);
  Serial.print("Self Test:     0x");
  Serial.println(self_test_results, HEX);
  Serial.print("System Error:  0x");
  Serial.println(system_error, HEX);
  Serial.println("");
  Serial.println("------------------------------------");
  delay(500);
}
```

Fig. 12. The IMU sensor #1 data display code.

Code line 41-43 are using the Commander methods to finish defining the user interaction commands taken by the program via the serial monitor. The *command.add()* method takes a char argument as a letter or word, a reference function that links to the previous argument and ends with an explanation label of type string.

The main loop of the program contains the built-in *move()* function that generates motor rotation according to user input and *run()* method that keeps open the interaction with the user (*code line 57-60*).

Setting up the *Adafruit IMU* sensor is relatively simple (Fig. 11 and Fig. 12). Most of the data access is given by the developer libraries. The algorithm to convert raw data to actual space coordinates is fusion-integrated into the chip (*code lines 1-4*). The sample rate is a trial-and-error setup, and 100 samples per millisecond seem like a decent starting point, given the high accuracy of the chip (*code line 7*).

Code line 10-11 define the instance for each sensor, allocating IDs and the memory address.

An important aspect to know is that all *BNO055 sensors* are sold with the same ID and memory address. These details need to be manually changed to avoid data collisions.

The *displaySensor_1* and *displaySensor_2* functions are generally identical, and they define each sensor object and link their memory allocations (*code line 16-17*). The rest of the line of code from Fig. 11 helps display the identification data, the sensor data and the error data in the serial monitor window.

The *displayCalStatus()* method from Fig. 12 is an essential self-calibration technique that allows the sensors to auto-correct their data in case of system errors. This function also helps program the sensor to reset the data generated at the system start-up or link interrupted data streams if data pockets correlate (Fig. 13-15).

```
/* Display sensor calibration status */
/**************************************************************************/
void displayCalStatus(void)
{
  /* Get the four calibration values (0..3) Any sensor
  /* 3 means 'fully calibrated" */
  uint8_t system, gyro, accel, mag;
  system = gyro = accel = mag = 0;
  bno_1.getCalibration(&system, &gyro, &accel, &mag);
  bno_2.getCalibration(&system, &gyro, &accel, &mag);

  /* The data should be ignored until the system calibr
  Serial.print("\t");
  if (!system) { Serial.print("! "); }
}
```

Fig. 13. The IMU autocalibration function.

```
void setup(void)
{
  Serial.begin(115200);
  Serial.println("Orientation Sensor Test"); Serial.println("");
  /* Initialise the sensor */
  if(!bno_1.begin() || !bno_2.begin()) {
    /* There was a problem detecting the BNO055 ... check your co
    Serial.print("1 or 2 x BNO055 NOT detected ... Check your wir
    while(1);
  }
  delay(1000);

  /* Display some basic information on this sensor */
  displaySensor_1();
  displaySensor_2();

  /* Optional: Display current status */
  bno_1.setExtCrystalUse(true);
  bno_2.setExtCrystalUse(true);
}
```

Fig. 14. The IMU set-up.

```
void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event;
  bno_1.getEvent(&event);

  /* Display the floating point data */
  Serial.print("     "); //X: ");
  Serial.print(360 - event.orientation.x, 0);

  bno_2.getEvent(&event);

    /* Display the floating point data */
  Serial.print("     "); //X: ");
  Serial.print(360 - event.orientation.x, 0);

  /* New line for the next sample */
  Serial.println("");
```

Fig. 15. The IMU main loop.

The setup function sets the serial monitor baud rate to stream the data, checks the existence of each sensor and alerts in case of non-detection and displays the identification information of each sensor (*code line 83-101*).

The main loop creates a continuous data stream event for each sensor, reverses the angle orientation by subtracting from 360 degrees the values measured by the sensors and prints the data in the Arduino serial monitor (Fig. 14, *code line 106-123*).

The aforementioned Arduino code is also available on GitHub at the following address: github.com/deemano/IMU_AMT_sensing.

6. Results

Controlling the velocity and position of the *Brushless DC Motor* has been a great learning experience and a success.

The main concern was the lack of processing power in the *Arduino Mega* to be able to serve the entire robotic system requirements, including data feedback from both *IMU* sensors and data generated by the motor encoder and the shield. Even if, by the end of this research, the *Arduino Mega* could centralize well all the data, because of its size and weight, it is impossible to integrate into the robot structure. Going further into the next phase of the project, another more robust, light weight and powerful microcontroller is necessary.

Moreover, changing the microcontroller will also require changing the *BLDC Controller* with something much lighter.

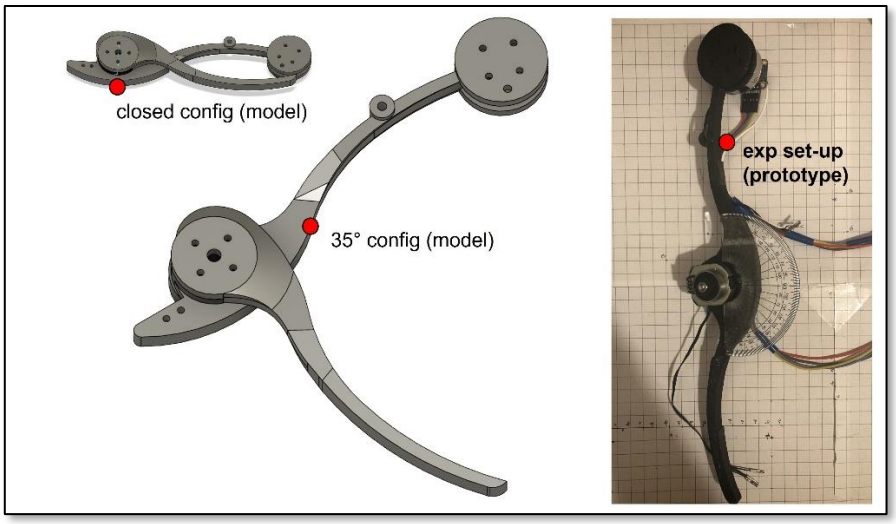

Fig. 16. Preliminary test of the angular 'knee' configuration of the model (left panel) and of the real prototype (right panel).

The IMU sensors remain an excellent asset for future development, providing accurate data in a compact fused shape and format.

The *AMT103 Rotary Encoder*, on the other hand, was only an option because of its compatibility with the *Simple Foc Shield*. It is a heavy device, noisy and inaccurate. A series of tests not mentioned in this paper have been done with the *AMT223-V Absolute Encoder*, and the results prove it could be one of the best options for the next phase of robot development.

The actuators failed the tests because they did not offer enough torque, and they were unreliable. An issue with one of the motors was an offset deviation around the shaft axis; this problem was difficult to debug and created a chain reaction of junk data captured by the sensors.

At this stage an analysis of the angular excursion of the modelled design and of the manufactured prototype has been developed (Fig. 16): the main articulation joint (i.e. the 'knee') angular displacement was simulated on the modelled design and then on the 3D printed prototype by means of a goniometer.

The design of the robot will need more testing, but the frame structure is an aspect that will need to be re-evaluated, since current testing and validation are in a preliminary stage. Moreover, these results will require a proper comparison with current performance of monoped robots, such as, for example, Salto [8] and Ascento [10].

7. Conclusion

This project has been focused on gathering the necessary knowledge and data to build an omnidirectional monoped self-balancing jumping robot.

The evaluation and testing of different equipment combinations, control methods and programming algorithms accumulated during this work will be of great value to help push this project into its next development stage. However, at this stage, further research can be explored in order to refine the design in view of the effective geometry and kinematic of the robot [17] by also taking on board bio-mimetic and biologically inspired principles which could make this design inherently optimized [18-21].

High accuracy control, digital simulation, system modelling, energy efficiency and intelligent design are just a few significant challenges of the future phase.

Supplementary Materials

The following supporting information can be downloaded at: github.com/deemano/IMU_AMT_sensing.

Author Contribution

All authors contributed to this paper. DM developed the robot design, implemented the HW and SW and performed the integration. ELS supervised and refined the paper preparation. All authors read and approved the final paper.

Funding

This research was kindly funded by the Pro Vice-Chancellor (Research) UG Summer Scholarship Scheme, Liverpool Hope University.

Acknowledgment

This work was presented in report form under the supervision of EL Secco in fulfilment of the requirements for the Beng in Robotics for the student Vasile Denis Manolescu from the Robotics Lab, School of Mathematics, Computer Science & Engineering, Liverpool Hope University.

Conflicts of Interest

The authors declare no conflict of interest.

Appendix

This appendix reports some additional details of the project. Fig. 16 shows a prelimanryt set-up for testing thre motor speed and the angle testing combined with the BNO055 testing and calibration equipment. Fig. 17 reports some details of the opened actuator, whereas Fig. 18 displays the testing of the *Simple Foc shield* with the Arduino board. Finally, Fig. 19 presents the assembly of the Monopod robot with a screenshots of the standing test at 90 degrees (Fig. 20).

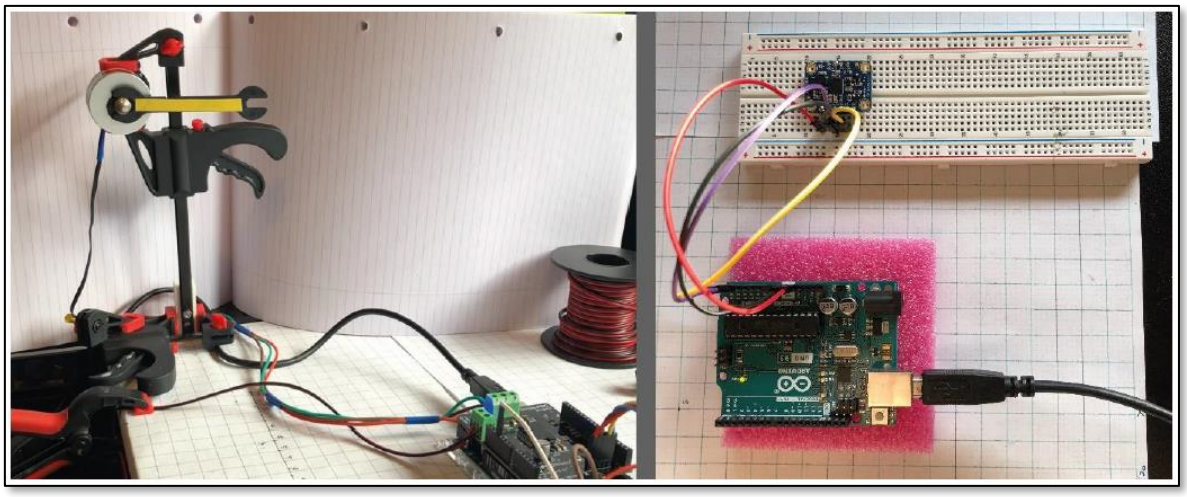

Fig. 16. The Motor speed & angle test (left panel) and the *BNO055* testing and calibration.

Fig. 17. The dismantled actuator.

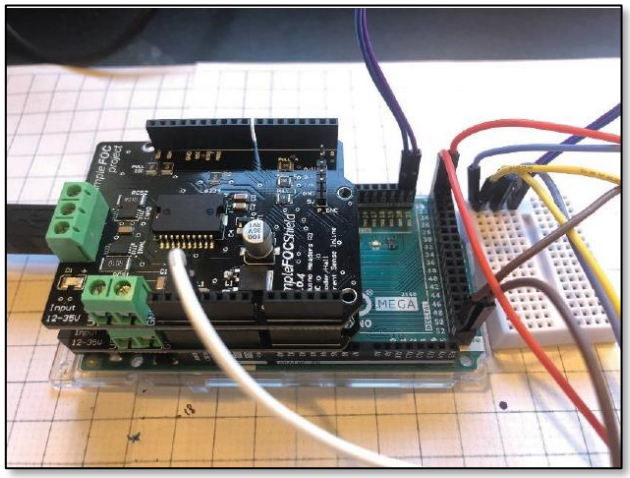

Fig. 18. Testing the *Simple Foc shield* with the Arduino board.

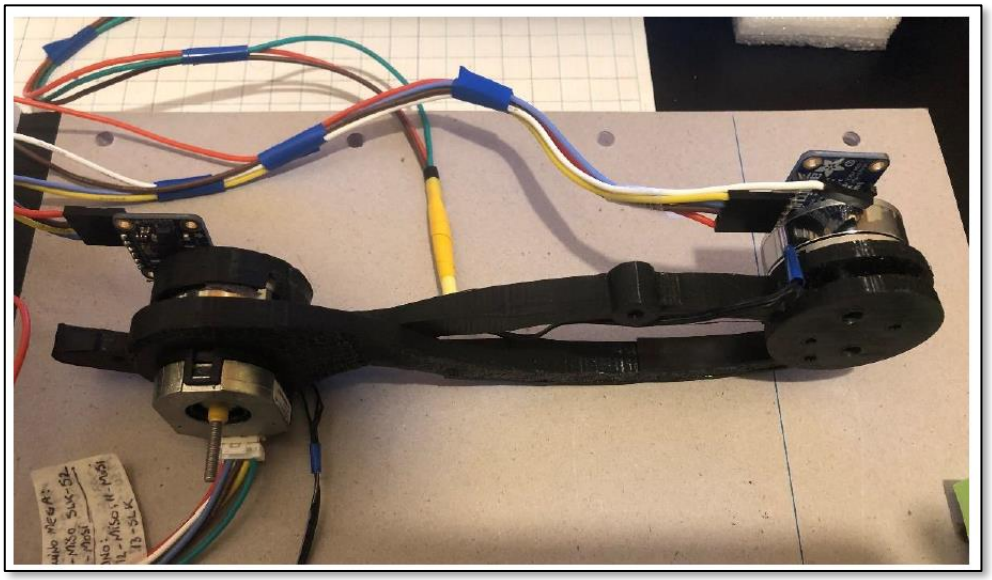

Fig. 19. The fully assembly of the Monopod Robot

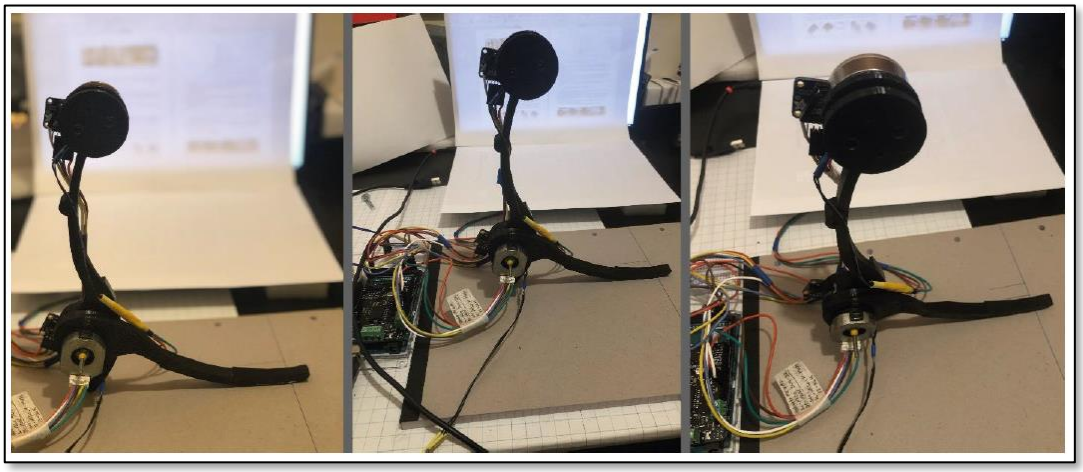

Fig. 20. Standing test @ 90 degrees.

References

[1] NASA, H. S. S., 1967. Lunar Pogo Stick. [Online] Available at: https://arc.aiaa.org/doi/10.2514/3.28991 [Accessed 2022].

[2] NASA, R. D. M. K. e. a., 1971. The Lunar Hopping Transporter. [Online] Available at: https://ntrs.nasa.gov/api/citations/19730007490/downloads/19730007490.pdf [Accessed 2022].

[3] Yim, J., 2020. Hopping Control and Estimation for a High-performance Monopedal Robot, Salto-1P. [Online] Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-108.pdf [Accessed 2022].

[4] Matteo Rovatti, N. A., 2017. A GUIDE TO QUADRUPEDS' GAITS. [Online] Available at: https://www.animatornotebook.com/learn/quadrupeds-gaits [Accessed 2022].

[5] Justin K. Yim, R. S. F., 2016. Precision Jumping Limits from Flight-phase Control in Salto-1P. [Online] Available at: https://people.eecs.berkeley.edu/~ronf/PAPERS/jyim-iros18.pdf [Accessed 2022].

[6] Xiaojuan Mo, e. a., 2020. Jumping Locomotion Strategies – From Animals to Bioinspired Robots. [Online] Available at: https://www.mdpi.com/2076-3417/10/23/8607/pdf-vor [Accessed 2022].

[7] P. Saraf, A. Sarkar and A. Javed, "Terrain Adaptive Gait Transitioning for a Quadruped Robot using Model Predictive Control," 2021 26th International Conference on Automation and Computing (ICAC), 2021, pp. 1-6, doi: 10.23919/ICAC50006.2021.9594065.

[8] J. K. Yim, B. R. P. Singh, E. K. Wang, R. Featherstone and R. S. Fearing, "Precision Robotic Leaping and Landing Using Stance-Phase Balance," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 3422-3429, April 2020, doi: 10.1109/LRA.2020.2976597.

[9] A Bipedal Walking Robot that Can Fly, Slackline, and Skateboard," Science Robotics (AAAS), Vol. 6, Issue 59, eabf8136, 2021, doi: 10.1126/scirobotics.abf8136.

[10] V. Klemm et al., "Ascento: A Two-Wheeled Jumping Robot," 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 7515-7521, doi: 10.1109/ICRA.2019.8793792.

[11] DW. Haldane, MM Plecnik, et al., "Robotic vertical jumping agility via series-elastic power modulation", Science |Robotics, 1(1), doi: 10.1126/scirobotics.aag204.

[12] Doug M. Boyer, E. R. S. J. T. G. J. I. B., 2013. Evolution and Allometry of Calcaneal Elongation in Living and Extinct Primates. [Online] Available at: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0067792s [Accessed 2022].

[13] VD Manolescu, EL Secco, Design of an Assistive Low-Cost 6 d.o.f. Robotic Arm with Gripper, 7th International Congress on Information and Communication Technology (ICICT 2022), Lecture Notes in Networks and Systems (ISSN: 2367-3370), 1, 39-56, 2022.

[14] VD Manolescu, EL Secco, Design of a 3-DOF Robotic Arm and implementation of D-H Forward Kinematics, 3rd Congress on Intelligent Systems (CIS 2022).

[15] MIT, M. &., 2021. Arduino SimpleFOCShield v2.0.4. [Online] Available at: https://docs.simplefoc.com/arduino_simplefoc_shield_showcase [Accessed 2022].

[16] Townsend, K., 2022. Adafruit BNO055 Absolute Orientation Senso. [Online] Available at: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor [Accessed 2022].

[17] V. Kumar, P. E. U. o. P. S. o. E. a. A. S., 2016. Introduction to Robot Geometry and Kinematics. [Online] Available at: https://www.seas.upenn.edu/~meam520/notes02/IntroRobotKinematics5.pdf [Accessed 2022].

[18] Amanda Sutrisno, D. J. B., 2020. How to run 50% faster without external energy. [Online] Available at: https://www.science.org/doi/10.1126/sciadv.aay1950 [Accessed 2022].

[19] Magenes G, Ramat S, Secco EL, (2002), Life-like Sensorimotor Control: from Biological Systems to Artifacts, Current Psychology of Cognition, Vol. 21(4-5), pp. 565-596.

[20] EL Secco, C Moutschen, TF Agidew, AK Nagar, A sustainable & biologically inspired prosthetic hand for Healthcare, EAI Transactions on Pervasive Health and Technology, 4, 14, 2018.

[21] D Campbell, EL Secco, Design of an Omni-Direction Robot with Spherical Wheels, International Journal of Engineering (IJE), 14 (1), 2022.