Autonomous and Remote control Biped

A project based on the self-balancing Biped

Mahendra Kodavati
Computer Science and Electrical Engineering
University of Illinois, Urbana Champaign
Champaign, USA
mk102@illinois.edu

Rohan Nedungadi
Computer Science Engineering
University of Illinois, Urbana Champaign
Champaign, USA
rohanrn3@illinois.edu

***Abstract*—Autonomous and Semi-Autonomous systems are increasingly going to become important for humans in order to be able to explore beyond their natural physical restrictions. Enabling vehicles and systems to be able to operate without a human in the loop (or at least one that is not physically present), allows us to go beyond the previously seen levels of explorations. We are able to build more capable systems that could operate in environments unsuitable for human beings and also operate longer continuous hours without having the need to rest or replenish energy supplies as frequently and as much as humans require. Hence, in this paper we introduce a lightweight, low cost embedded system design for a biped (Fig 3), that can be remotely controlled and also perform fully autonomous tasks with feedback using computer vision and time of flight sensors. A system such as this could benefit studies such as out of space exploration, where the cost of sending a human being is exponentially higher than a robot . A device such as this can be either remotely controlled or set to perform certain autonomous tasks without having a human to overlook it.**

***Index Terms*—autonomous, biped, computer vision, remote access**

Fig. 1. ESP32

I. Introduction

With the help of efficient and powerful embedded systems and equally sophisticated systems designs, it is capable to build cost-effective but powerful systems. For this project with the ESP32's (Fig 1) dual-core chip and with the help of feedback generated from the sensors on the biped, we were able to design a control loop to let the Biped balance itself on two wheels while also performing other actions as directed.

Identify applicable funding agency here. If none, delete this.

Fig. 2. FreeRTOS

Additionally, in order to make the biped autonomous/semi-autonomous the Wi-Fi chip on the ESP32 and the camera module on the biped were leveraged, to push data to an external server for remote control and computer vision processing.

All put together, we designed a self-balancing biped robot that could perform the following tasks :

- Be remotely controlled over the WiFi from a mobile app that displays the camera feed and buttons
- Use computer vision remotely to perform object detection and carry out autonomous activities

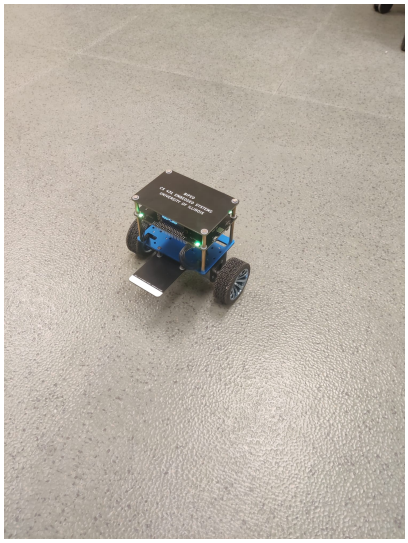

Fig. 3. Biped

II. System Design

A. Hardware

The bipeds range of sensors included the IMU, motors (encoders) in order to perform the basic actions. Additionally, the design incorporated a Display screen for outputting information and Neo Pixel LEDs to perform some kind of indications. Furthermore, the system on board the Biped consisted of push buttons, time of flight sensors and a camera, which could be used to take input from the users. Since the Esp32 does not have sufficient pins to communicate directly to all the sensors, we also had the IO expander on board.

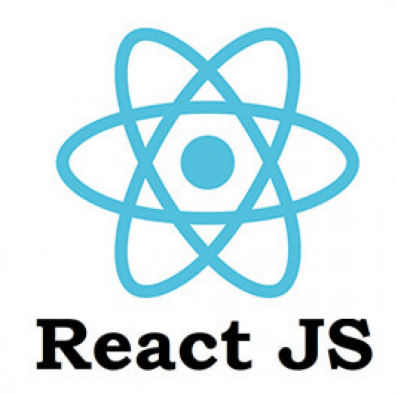

Fig. 4. ReactJS

B. Embedded Software

FreeRTOS [4] (Fig 2) was used to create tasks which could be planned and scheduled using GRMS concepts. Additionally, we used control theory to create a control loop with feedback from data collected from the several sensors/tasks to actuation using the motors/encoders.

C. Autonomous and Remote software

Remote access and camera streams were generated using the HTTP framework, (Fig 7). The mobile (webapp) app (Fig 8) was built using ReactJS (Fig 4) that created HTTP triggers to the biped. Object detection and recognition was performed using YOLOv5 (Fig 5) by ultralytics [**?**]. Visualizations are done in python using the OpenCV framework.

Fig. 5. YoloV5 object detection

D. Architecture

For this project we had an additional device which hosted the front-end ReactJS server to render the UI, and as well as run the object detection inference on the camera stream. As seen in Fig 4 the UI running on the phone pulls the camera stream and can trigger HTTP requests on the port 80 server (Fig 6).

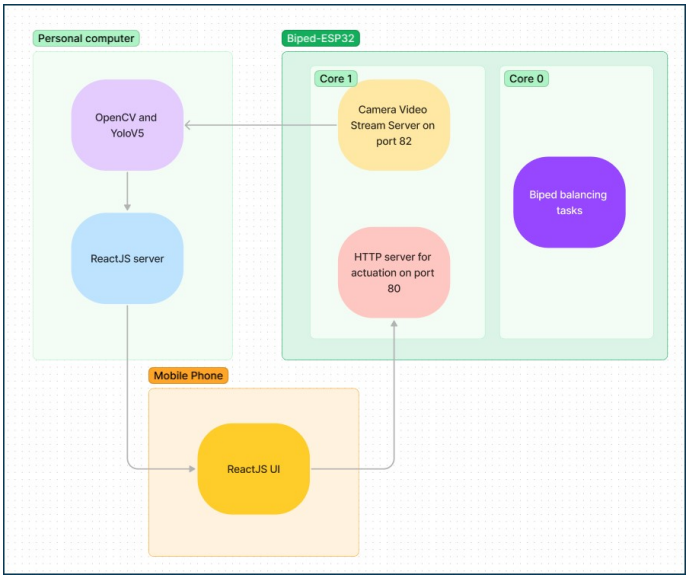

Fig. 6. Architecture

III. Implementation

First, we began with streaming the camera frames using an HTTP server hosted on the ESP32 module. For this we had to have an independent server completely for the camera. To add upon this, we added 5 more RESTful HTTP endpoints, as shown in table1. These HTTP endpoints (Fig 9) would control the biped in a certain direction by creating a waypoint plan for the biped to follow in that direction. These end points were then attached to the React JS front end, which had individual buttons to trigger each of these end points, as seen in Fig 9. The ReactJS front end also held a continuous HTTP request to stream the camera frames to the app. This combination allowed the user to operate the biped from a different room/place as long as they were able to connect to the biped over the internet.

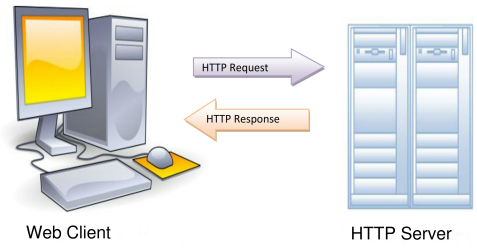

Fig. 7. HTTP Server

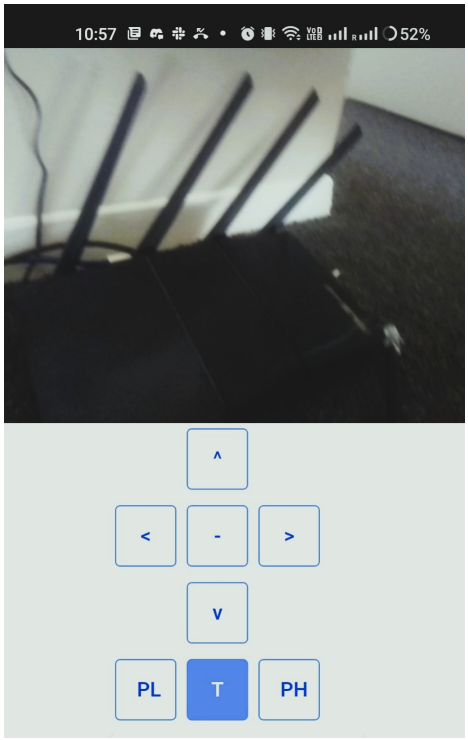

Fig. 8. Mobile app to control the biped

IV. Challenges

A. HTTP server

Since the ESP32 is more of a microcontroller, it does come with extensive support for parallel processing and large memory buffers. Hence, the HTTP server design is that of a lightweight C++ server with limited memory and threading capacity. This means the HTTP server is bound to a single port and a single user accessing that port at a time, i.e. if the server is servicing a single request, it won't be able to service other requests until it has completed serving the older one. That mean newer requests will be lost until the server gets free from the previous request. Since the camera stream is a single continuous, request we will not be able to trigger the other endpoints by just hosting a single server for camera stream and actuation. Hence, we had to create two different independent servers on two different ports (namely 82 for stream and 80 for web requests).

B. Camera Streaming and Actuation

Furthermore, due to the limited memory on the board, camera frames would be buffered two at a time at most (as the heap memory is not sufficient to run other parallel tasks along with it). Upon adding the additional HTTP server for actuation, the streaming application would crash and not be able to provide a valid response to stream requests. Upon some debugging, we realized that the board was running out of memory, and so the alternative we tried was to reduce the stack space provided to all other tasks sufficiently enough to not affect our application from failing but also allowing streaming

Endpoint	Port	Description
/camera	82	continuous, open connection to stream the camera video
/forward	80	trigger the biped to move forward using a waypoint
/park	80	trigger the biped to stop at its current position
/reverse	80	trigger the biped to move backward using a waypoint
/left	80	trigger the biped to turn left using the imu
/right	80	trigger the biped to turn right using the imu

Fig. 9. HTTP trigger points

to work. This solved the problem, and we were successfully able to perform actuation while streaming the video frames. One additional issue that arose after this was latency. Since all the tasks including the servers were running on a single core, context switching was causing a small delay, and so to avoid it, we shifted the camera RTOS task with both the servers onto the second core, making the application more stable and robust.

V. Discussion and Future work

A. Single Stream

Since the camera allows us to only stream a single stream, at a time, to have remote access with the computer vision was challenging at first. We were able to pull the stream in python with OpenCV and perform object detection and then attempted to push the stream to the web app. Unfortunately, this was leading to slow frames and unseen memory issues. Since the HTTP server on the biped for the stream is a lightweight version, it does not handle failures and broken connections well, which was creating a problem. Hence, in the future, we could explore more on the HTTP package and look into how we could stabilize the system.

Fig. 10. TinyML framework on ESP32

B. Running Computer Vision Locally

With newer advancements, we see that popular machine learning framework developers have come out with smaller

and efficient architectures could be run on embedded systems. The official package TinyML [3], by TensorFlow, is currently capable of running on the esp32 and is not bad in terms of performance. This framework essentially enables us to convert tflite models which are highly optimized models with the help of quantization and pruning to reduce model sizes. While we have not exploited the details of what this framework is capable of in entirety, it would be interesting to see its use cases and future advances for the ESP32 board.

Fig. 11. Trained YOLO object detection model

C. Complex interactions

Additionally, we were designing object detection models and training the biped to identify other biped, making it similar to autonomous cars. Due to shortage of time, we were unable to fully present it, but believe it is an interesting avenue for future work on the biped for a project (Fig 11).

VI. Conclusion

In this project, we extended the capabilities of the biped system by providing remote access and using computer vision to enable it to make autonomous decisions. With the help of the Wi-Fi module on the ESP32 and with some additional memory management tuning, we are able to successfully deploy HTTP servers on the biped to stream the camera and as well as actuate the system with the help of a mobile app. This project aims to present the capabilities of the biped for use in AI driven projects, even with its embedded architecture that does not consist of sophisticated hardware. We hope that this project paves the way for many more useful applications and projects going ahead.

Acknowledgment

We would like to thank our Course Instructor Prof. Lui Sha for providing us with material to understand embedded systems, real time system designs and giving us the opportunity to build this bot for our coursework. We would also like to thank the TAs for the course Simon Yu and Haoqing Zhu for designing and assembling the hardware for us to utilize to work on this project.

References

[1] https://github.com/ultralytics/yolov5
[2] https://react.dev/
[3] https://www.tensorflow.org/lite/microcontrollers
[4] https://www.freertos.org/