VIERBEINER DEVELOPMENT DOCUMENTATION

"The QuadBot emerged due to a need of making a relatively complex project in order to get equipped with the skills required to make my CV more impeccable. The possibility of applying for an internship in my third semester made me realise how excruciatingly lacking my skillset is. This led me to change my view from "learning the skills and then start a project based on the things I know" to "start a project and learn all the skills needed to complete it"."

Description:

Vierbeiner is a four-legged mobile robot controlled by a radio controller. Designed and developed from scratch, Vierbeiner is a project implemented to aid learning. It has three degrees of freedom a horizontal plane. What makes it unique from the general IoT based 4-legged robots is that it uses a dual-leg algorithm opposed to the single-leg algorithm. This gives Vierbeiner more speed and efficiency.

Inception:

Vierbeiner is a project which aims to build a four-legged mobile robot. The first phase of the project aims to make the robot mobile with the help of a controller. The second phase contains different forms of control, using environmental sensors, or hand gesture recognition algorithms. The third phase will see Vierbeiner become autonomous, and rely solely on itself for control. Although the idea was perceived at the beginning of the year 2024, active development did not start until the 30th of October, 2024. The first phase of the project was completed on the 5th of December, 2024. However, at this point, further development towards the next phases was deemed unlikely to proceed.

Electronic Design:

Vierbeiner is powered by a 6V source. A 2S 2200mAh battery along with a stepdown convertor was used as the power source. They powered a circuit board which had a common supply and ground to which the MG90S motors and the Arduino Mega were connected. The analog pins of the motors were also connected to the analog outputs of the Arduino Mega through the same perfboard.

Software Design:

Vierbeiner was coded using C++ using the Arduino IDE and its library. Object-Oriented Programming concepts like Polymorphism and Abstraction were extensively used. Use of functions and variables for their own use made change of parameters and development of new movement algorithms simpler and faster.

Structural Design:

Initially wanting to 3D print a custom frame, the connecting joints were purchased from a third-party seller. For the base that gives the robot stability to carry out the dual-leg algorithm, header pins were used due to their low surface area and long reach across the ground. Epoxy putty was used to make the moulds which were placed on the header pins, with the moulds having holes that fit the legs. This design gave the base just enough friction to not only push against the ground, but also low enough allow the base in contact with the ground to change the distance between each other during movement.

List of components used:

1. Arduino Mega 2560
2. MG90S motors
3. 2S 2200 mAh LiPo Battery (with XT60 connector)
4. Stepdown convertor
5. Plastic frame
6. 20 AWG wires
7. Jumper wires
8. Double-sided perfboard
9. Switch
10. nRF24L01 Radio Module (Communication)
11. Unseparated Header Pins
12. Epoxy Putty
13. Arduino Uno (Controller)
14. Joysticks (Controller)
15. Breadboard (Controller)

Development Log:

30.10.2024

One of the major problems of making this robot from scratch was the structure. Initially having no clue regarding how to proceed, I researched about 3D printing. I decided to learn Fusion 360 in order to model the frame and have it printed. Since this was not an easy task, I found another alternative – I found pan and tilt mounts used for cameras, and decided to use those as the primary mobile parts. However, it was an unesthetic approach, and the problem of the end limbs was still present.

31.10.2024

I realised there were ready-made frames available on the internet. There was only one available 12 DoF frame available, which was purchased. The following components were also purchased:

- Arduino Mega
- Arduino Uno

- x12 SG90 Servo Motors
- Bluetooth Module
- Breadboard

The Arduino Mega was chosen to control the robot due to the sufficient availability of PWM pins.

02.11.2024

The design of the power supply was realised, and the following components were ordered:

1. x2 2S 2200MaH 35C LiPo Batteries
2. 14 AWG Wire
3. Potentiometers
4. Step Down Convertor
5. Perf-board

Each servo motor needs 5V supply, and the stall current is 0.75A. This means the maximum current that can be drawn is 9A. Due to this, I could not power the motors through the microcontroller or the breadboard. Therefore, 14 AWG wires, a voltage regulator and a perf-board were ordered to have an external power supply for the motors. To power the motors, the 7.4V supply from the battery is stepped down to 5V and connected on the perf-board in parallel to 12 supply lines leading to the motors. The ground lines are connected on the same perf-board back to the battery. The Arduino is connected directly to the battery, and the voltage is stepped down by its built-in regulator.

03.11.2024

Designed the structure of the code and modelled the basic kinematics of the robot.

1. Servo Delay movement
2. Function Declarations
3. Derived Servo Class for custom data members

Realised that the wrong type of servo was ordered, cancelled order and ordered MG90 motors.

04.11.2024

1. Solved the motors' basic multi-step problem. Several motors can now be moved at the same time, at a controlled pace.
2. Standardised the orientations of all motors.
3. Solved the rotational, translational and redundant kinematics.
4. Forward algorithm is completed.
5. Ordered new components.
6. Changed control transmission medium, from Bluetooth to radio.

7. Devised a kill switch operation.

For the forward algorithm, robot can be, at any point, performing on of the three functions: stepRight(), stepLeft() and stance() (Excluding rotation() because it has not yet been defined).

Here’s what stepRight() does:

1. Moves translational motors 0 and 2 up if they are not up
2. Moves rotational motors to stepRight() threshold
3. Moves translational motors 0 and 2 down
4. Moves translational motors 1 and 3 up
5. Moves rotational motors to stance state (translational motors 1 and 3 are still up)

stepLeft() follows the same steps as above, but of course towards the inverse counterparts.

Since the rotational motors are at stance state at the end of stepRight() and stepLeft(), stance() only lowers any translational motors that may be up. To make sure that the robot is in stance state, all motors will be updated to match their stance angles.

Due to the way in which stepRight() and stepLeft() reach a semi-stance state at the end of their executions, all state transitions between the three forward algorithm functions can be achieved smoothly.

Orientations of the motors are as follows:

- Orienting front-right rotational motor to 0 at “closed, pointing right”.
- Orienting front-left rotational motor to 0 at “open, pointing right”.
- Orienting hind-left rotational motor to 0 at “closed, pointing left”.
- Orienting hind-right rotational motor to 0 at “open, pointing left”.
- Orienting the translational motors to 0 at “stretched outwards, pointing sideways”.
- Orienting the redundant motors to 0 at “closed inwards, pointing upwards”.

Components that were ordered:

1. Kill switch (controller)
2. 10cm Jumper wires
3. Radio module
4. Joysticks

05.11.2024

Made major development on code.

1. Standardised and isolated basic movements like rotation and translation. Using threshold() and functions like moveEvenUp(), total control was achieved which made making other movement algorithms way easier.
2. Made algorithms for movements (forward-backward, left-right, rotation) in 3 degrees of freedom on a 2D plane.
3. Completed all actuating parts of the code.

Only missing part of the code is the radio module for control. Code produces no errors, but detailed checking for logical errors is yet to be done.

07.11.2024

1. Created controller code
2. Setup radio communication code and joystick control.
3. Created first Diagonal movement algorithm.

08.11.2024

Ordered the following components:

1. High current switch for power supply
2. Header pins for perfboard
3. Gesture recognition sensor

09.11.2024 – 13.11.2024

1. Soldered the perfboard for the supply line. Changed the supply lines layout to directly fit analog, ground and supply pins of the motors along with the PWM output pins from the Mega.
2. Tested the hind rotational motors, which led to unexpected behaviour due to faulty code. Solution to the problem was, however, not found.

14.11.2024

Extensively tried setting up radio communication but failed.

16.11.2024

Tried setting up radio communication again, but failed. Searched procedure for Bluetooth communication, but did not work on it further.

19.11.2024

1. Patched the hind rotational motors' code. (Changed threshold values in threshold() parameters and replaced move() with threshold() itself.)
2. Conducted first successful test for above mentioned motors. Used local controller for testing.
3. Attached the hind rotational motors to the frame.

20.11.2024

Realised that operating current of the MG90Ss was 120-250 mA, and at any moment, a maximum of four motors would be operating. Therefore, at the point where maximum power was being drawn, there would only be 1A of operating current required along with 80 mA of 8 stationary motors (10 mA each when stationary) and maximum of 600 mA through the Mega. Therefore, it is unlikely for the current to exceed even 3A.

1. Changed supply parameters: 20 AWG wires and 3A rated convertor were deemed enough for transfer of power.
2. Tried to configure the convertor once again, but ended in another frustrating failure

21.11.2024

1. Successfully configured the convertor.
2. Soldered the entire supply line.
3. Tested rotational motors using the supply line.

22.11.2024

Attached the translational motors. An error in the translational motor kinematics was found. The designed behaviour suited only the right translational motors and not the left. Therefore, their movement kinematics were redesigned. Left and right translational motors now have different threshold values.

24.11.2024

Attached redundant motors. All motors were tested and are functioning perfectly. The entire frame has been assembled. Attached the convertor and perfboard under the frame using adhesive double tape. Left redundant motor kinematics were changed to compliment the new changes.

A problem regarding the structure has arisen – some of the screws get loose quickly and have to be tightened every few minutes. Another worse problem is that each joint is not perfectly firm in terms of connections, and the plastic tends to bend under stress. These tiny errors add up to cause an overall bend when the robot stands on its own weight.

Planned on using connected header pins as a base, using epoxy putty to connect I perpendicular to the foot.

27.11.2024

Tried fixing the frame issues with no avail. Replaced the motors by other motors in translational and redundant positions, considering the possibility that the problem was with motor threads. Although it did help up to a certain limit, the problems were still evident. Considered other solutions like applying super gel on the screw and nut contact, and applying putty in the rotational cap hole to keep it in place.

Realised that the small screws holding the rotational motors were getting loose due to the larger radius of the frame hole.

28.11.2024

Tried putting cardboard pieces in between the motor and frame connection so as to push the motor outwards and close the gap. Worked fine, but the cap hole issue for two rotational motors was still present, despite being reduced significantly. It has been planned to just connect the header pins base to lower the angular tilt that causes the cap to bend and get removed. Overall, there has been a significant improvement in the frame structure despite there still being a droop.

29.10.2024

1. Organised all the wires up to a certain limit
2. Implemented a switch for the robot which returned the loop function when switched off
3. Tested Vierbeiner for the first time

At first it was just moving about its own position and not pushing forward. This is was due to two reasons. First, the surface contact of the base of the legs with the ground was very small due to the tiny leg base. Because of this, there was not enough friction to push the robot forward. Second, the robot could not balance on two legs, so three legs moved while having contact with the ground, and since the legs were not raised, nothing happened.

I then held the two legs which were supposed to be in contact with the ground in position as they would be in the case of a stable base, and then the robot pushed forward. Vierbeiner walked for the first time, even if it was with assistance.

It is going to be quite a challenge to attach the base considering the nature of the point of contact. There has to be just enough friction to push the robot forward, but also not too much in order to let the distance between the points of contact with the ground decrease during movement.

30.12.2024

Made the putty mould for the legs. Instead of making a permanent connection between the legs and the header pins, I moulded the putty on the header pins such that the legs could just slide in and out.

1.12.2024

The mould was ready but the holes were too small to fit inside. Just made a temporary fix using tape, and Vierbeiner walked for the first time without assistance. Changed the translational up and down values so as to counteract the droop and not have contact with the ground, and the posture of the robot improved by a lot, now having more height and stability. Even the droop was fixed up to a certain limit. Made new moulds to fit the legs better.

3.12.2024

Made the controller using the radio module. Implemented the rotation algorithm.

Tried to get the gesture sensor to work for quite a while but did not succeed.

5.12.2024

Project Vierbeiner was complete. It walked flawlessly, controlled by the radio controller.