ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SEMESTER PROJECT REPORT

Omnibot: Mobile Furniture Baseline Development

Student: Chuanfang NING
Supervisor: Prof. Auke Ijspeert
Assistants: Dr. Anastasia Bolotnikova, Dr. Alessandro Crespi

7th January 2022

Contents

1 Introduction

1.1 Motivation

Helping people with limited mobility to enjoy the life as normal people do is always one of the main interests of robotic research. The efforts can be briefly summarized in 3 directions:

- Substitute: Help people regain part of their body function with neuroprostheics. (Zisis et al. 2021)
- Facilitate: Help people strengthen their weakened muscle with exo-skeleton. (Vouga et al. 2017)
- Rejuvenate: Train people to gradually reconstruct the connection between neurons with rehabilitation techniques. (D. Liu et al. 2018)

All of these approaches are universal approaches that are dedicated to increase the mobility of people from the basics. At a cost, they usually come with expensive devices with customization and calibration. And the patients themselves would also have to pay extra effort in learning to adapt to the devices and approaches.

A more economical way that could possibly benefit more people is that instead of directly improving the patients' mobility, we can render the infrastructures in the environments mobile to ease the life of patients. The project Omnibot is dedicated to such a solution: to drive the furniture around with a robot in a smart home assistive environment.

1.2 Project Goals

The overarching goal of the project is to design, implement and program the Omnibot for mobile furniture based on a pre-built omni-directional drive robotic platform that is shown in Fig. 1.

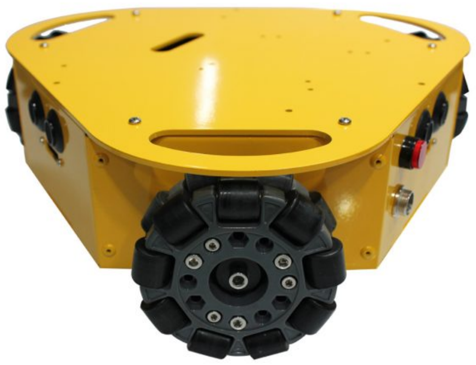


FIGURE 1: Nexus pre-built omni-directional drive platform

The pre-built robot platform with 3 omni-wheels and sonars is designed to run stand-alone on a low-end embedded system without any extensibility.

The project aims at extending this platform in mechanics, electronics and algorithm aspects to have a fully-functional robot that could render furniture mobile according to patients' needs. This robot would act as a baseline that is easy to replicate, extend and configure for future extensions in intelligent assistive home environment. The details of the project contents are described below:

- Mechanics
  - Design, implement and test the hardware attachment configuration of the robot to different types of furniture.
  - Extend the robot modules according to user demands (Bluetooth, LED)
- Electronics
  - Extend the stand-alone embedded system to an interactive embedded system allowing for module extensions and remote control.
  - Implement the teleoperation of the existing modules including motors, sonars and LEDs with high-level commands in ROS framework directly.
- Algorithms
  - Baseline localisation implemented with Optitrack in real-time.
  - Baseline navigation implemented with simplified visibility graph.
  - Baseline interactive control with predefined voice and gesture commands.
  - Baseline android mobile user interface application.

1.3 Report Outline

The remainder of this report follows the structure of three parts above:

Section 2 covers the mechanical extension, which explains 1) the attachment-configuration design, 2) the module extension and 3) future extension possibilities.

Section 3 covers the electronic extension, which explains 1) the limitation of the old system 2) the design of extended system and 3) detailed working process of sonars/motors/LED/Bluetooth.

Section 4 covers details about the algorithms for 1) localisation, 2) navigation, 3) interactive control with voice/gesture and 4) mobile application development.

Section 5 summarizes the overall implementation and makes claims from real-world tests.

The appendix section 6 includes the timeline of the project. And as supplementary materials, the internal Github repository (`https://ponyo.epfl.ch/students/mobfur`) includes all the source codes and instructions on how to set up the environment and use the software.

2 Mechanics

2.1 Attachment Configuration

For the hardware attachment, the goal is to propose and implement attachment configuration from a robot to a furniture in a secure, economic and interchangeable way. The attachment configuration is expected to have following functionalities:

- The attachment fully constrains the robot in motion without slips and play.
- The attachment doesn't hinder the normal use of robot and the furniture.
- The attachment configuration is interchangeable for different kinds of furniture.

Taking all demands into consideration, a "Tripous" attachment configuration came into design as shown in Fig. 2.

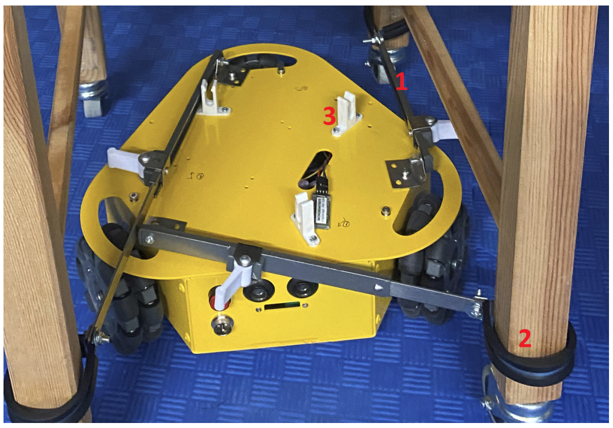


FIGURE 2: Attachment configuration "Tripous"

The design consists of 1) telescopic arms in range 230 368mm, 2) Hands made of R-Clamps with different sizes and 3) holder to rest the arm when not in use.

As its name implies, the "Tripous" will reach out its arms to grab the legs of the furniture and drive them as the main body of robot moves. The 3 arm connections ensure that the furniture is fully constrained and will not slip when it is driven by the robot.

The "Tripous" pushes and pulls the furniture around as robot moves. To reduce the friction, the furniture has to be equipped with passive wheels. The wheel is driven to resist any motion in still state to avoid the side effect of slipping brought by this design.

The arms are adapted from stainless steel window latches, which were designed to hold the window open in windy weather. Using window latches as arms would provide satisfactory mechanical strength of the connection while saving the cost of designing and manufacturing arms with high-strength materials.

The R-clamps are aluminum clamps that can be banded according to different furniture legs (round, square or rectangle). The R-clamps are mostly used in tube fixation and are robust against vibrations due to the cushioning rubber.

The telescopic arms and different sizes R-clamps allow the attachment configuration to fit into different furniture with different sizes and legs. The size of the furniture can vary as long as three of furniture's leg can fit in the ranges of 3 arms as shown in Fig. 3. And the legs of furniture can be any shape that fit into R-clamps in Fig. 4, which are available in different sizes (10~60mm diameter) from suppliers.

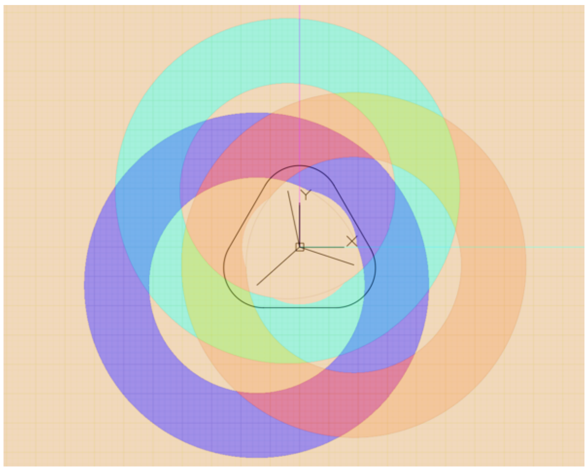


FIGURE 3: Telescopic range of arms

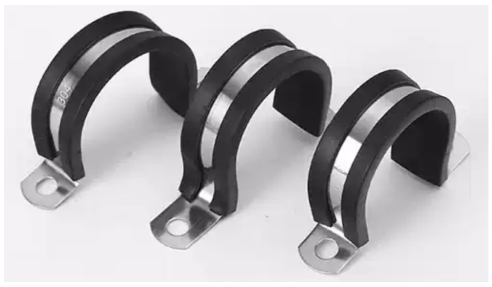


FIGURE 4: R-clamps

Fig. 5 and Fig. 6 show the simulated and real attachment configuration of Omnibot with a chair and a table with different sizes and leg shapes.

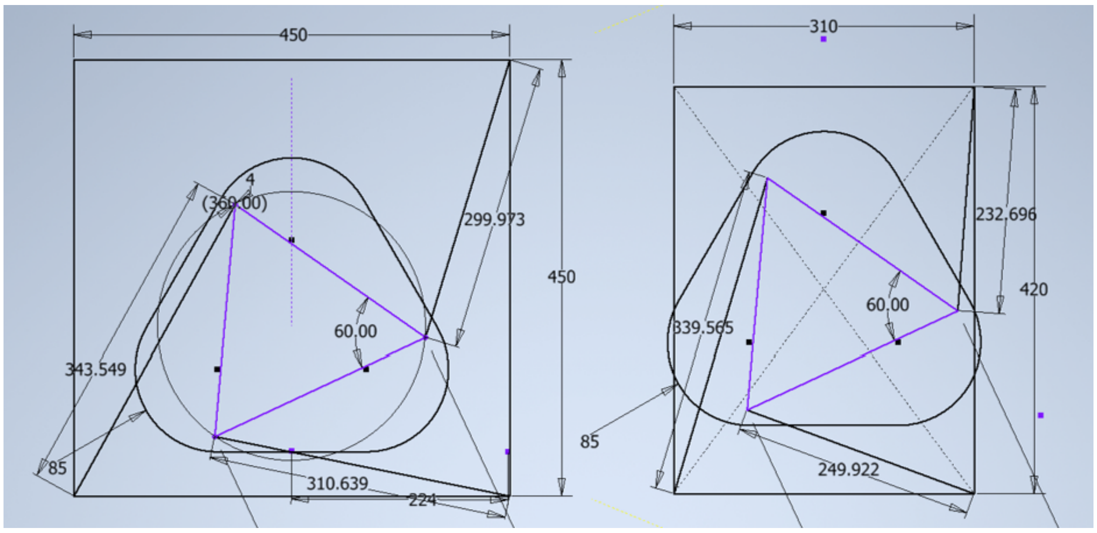


FIGURE 5: Simulated attachment configuration

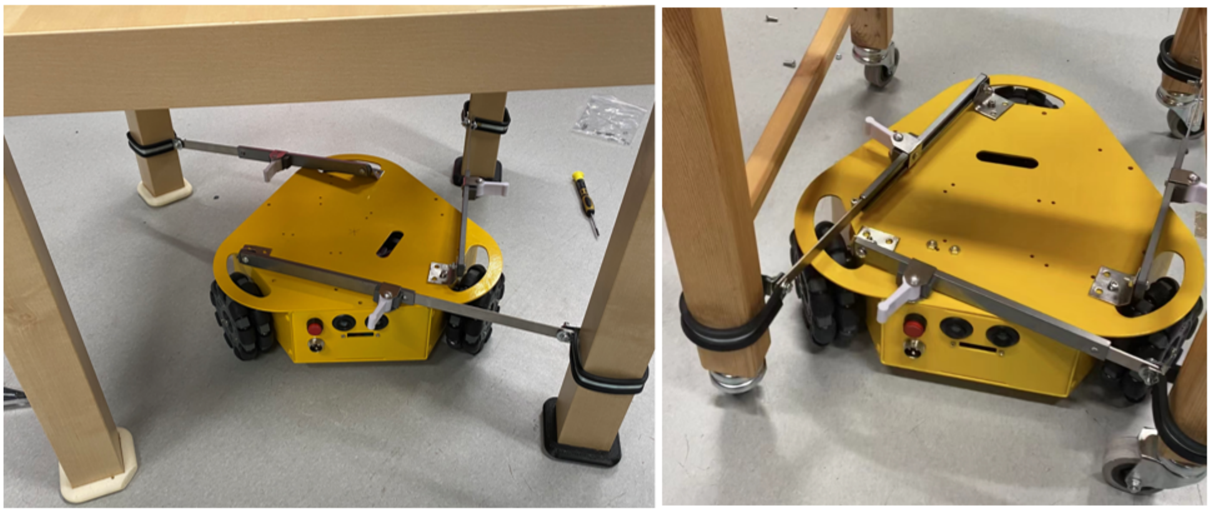


FIGURE 6: Real attachment configuration

One last trivial part in the "Tripous" design is storage clips for arms at still state as shown in Fig. 7. The arms would lock into the clip when pressed and could be released whenever needed.

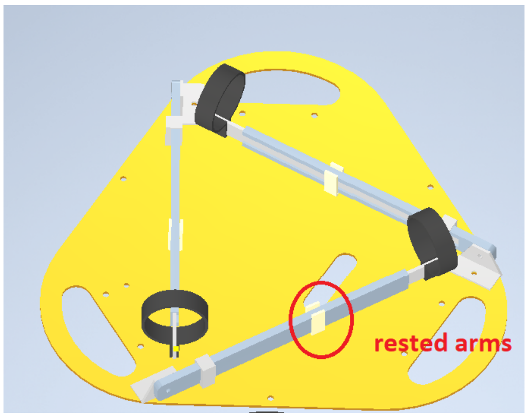


FIGURE 7: Attachment configuration "Tripous"

The proposed design mostly exploits products that were originally designed for other purposes, which makes the mechanical extension of Omnibot quite economic. The bill of materials (BOM) summed up to only 2CHF per set (shipping not included), the details of which can be found on the internal project archive.

2.2 Module Expansion

The Omnibot is expected to be extended according to user demands. In the project two extensions which do not require drastic hardware change were applied and a design to handle extensions requiring hardware change was discussed.

First, the Omnibot is extended with LED strip, which provides an intuitive indication of the operation and intention of the robot. A LED strip with 48 colored LEDs is pasted on the bottom of the robot platform. The LED strip shed light on the ground, from the reflection of which users can infer the states and intention of the robot. The LED is wired through a drilled hole of 32mm∅ to the on-board controller. The layout of the LED strip on the robot is shown in Fig. 8.

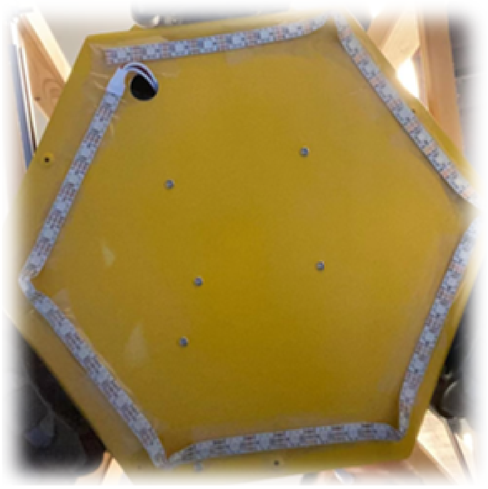


FIGURE 8: Layout of 48 LEDs on the hexagonal robot base

Secondly, the Omnibot is extended with a Bluetooth chip, which is more trivial in design. The bluetooth chip is exposed outside the metal shell to avoid being shielded.

It is also considered to extend the sensors of the Omnibot as the three sonars failed to provide enough information for the local obstacle avoidance. The space on the pre-built platform hardware is fully occupied and the extended sonars (lasers, infra or sonars) require extra space for placement. A possible solution to this issue could be to add a hardware support layer on top of the robot platform as shown in Fig. 9. The extended layer could be connected with the old top layer with printed ABS connectors. In the new design, the old layer would be used for a room accommodating extension modules and the new layer is connected with the "Tripous" robot arm structure.

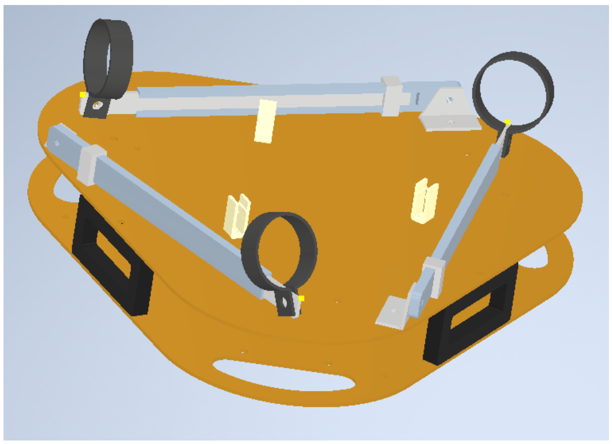


FIGURE 9: Design of the extra layer for more hardware module extensions

3 Electronics

3.1 Limitation of old system

The pre-built robot platform comes with a pre-configured embedded system with an AtMega-328P-20AU kernel as shown in Fig. 10. The board has an SP485-CN converter and an RS485 interface for sonars, an LS298SO20 motor driver powering at most 4 motors and an XbeePro wireless for wireless communication. However, the old system could be only configured to run the robot stand-alone with on-board processing of sonar data and motor commands. The board cannot be teleoperated as it doesn't provide enough Serial and timer resources for the sonars and teleoperation. The serial connection with different modules is switched physically via jumpers in the yellow circle. Also, the system does not allow module extensions since all pins are used either by the motor or the sonar.

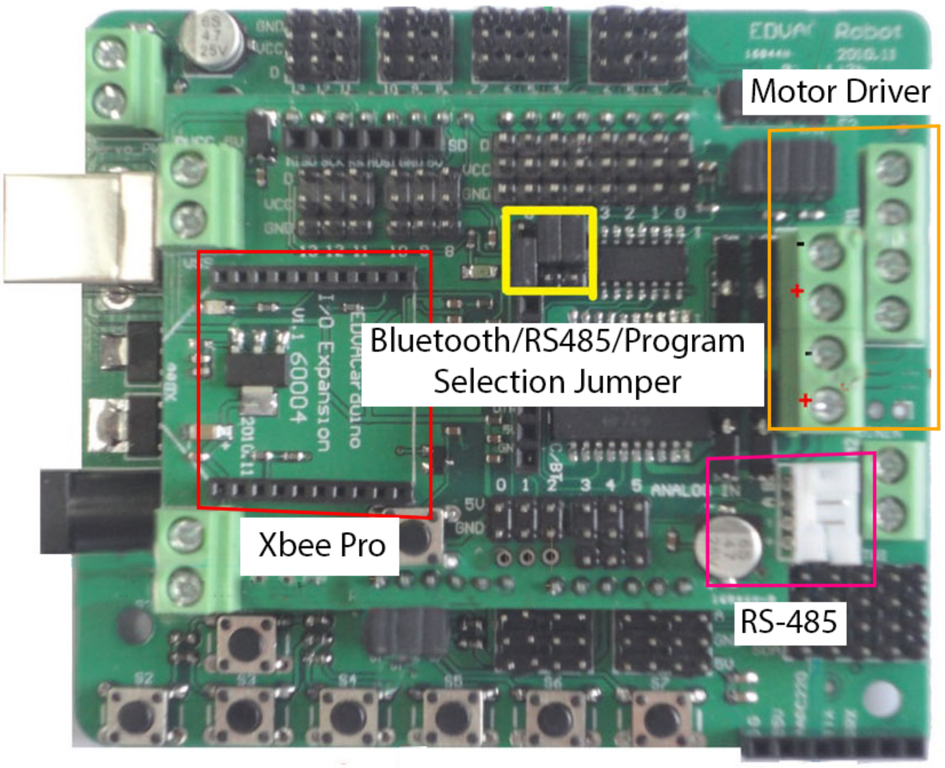


FIGURE 10: Default AtMega-328P-20AU embedded system

The Omnibot, however, is expected to run interactively according to user desires and with possibly more complex algorithms requiring more computing powers. What's more, Omnibot is expected to be extended with functionalities such as LEDs, new sensors or new actuators.

Due to the limitations of the old AtMega 328 system mentioned above, an Arduino Mega was introduced to increase the system capacity in the new electronics design, which sends the wheel speed commands to the old AtMega-328 system, provides new functionalities and allows the communication between the robot and the ROS masters. Tab. 1 shows the comparison of system resources in the old and new system. Although Arduino Mega has invested some resources on communicating with old AtMega-328 system, there are still a lot more resources available for more module extensions.

Resources	Old system	New system	
	AtMega-328	AtMega-328	Arduino Mega
Serials	Sonar	Mega	AtMega-328
	N/A	N/A	Sonar
	N/A	N/A	Bluetooth
	N/A	N/A	-
Timers	delay	delay	delay
	PWM pulses	PWM pulses	-
	PWM pulses	PWM pulses	-
	N/A	N/A	-
	N/A	N/A	-
	N/A	N/A	-
Digital Pins	14/14	14/14	17/54
Analogue Pins	2/6	2/6	0/16
PWM	4/4	4/4	0/15

[a] N/A: not existing [b] -: free for extension [c] a/b: used pins/all pins

TABLE 1: System resource comparison of old/new embedded systems

3.2 Connection plans

The connection plan of the whole new system with the AtMega-328 and the Mega boards is shown in Fig. 11.

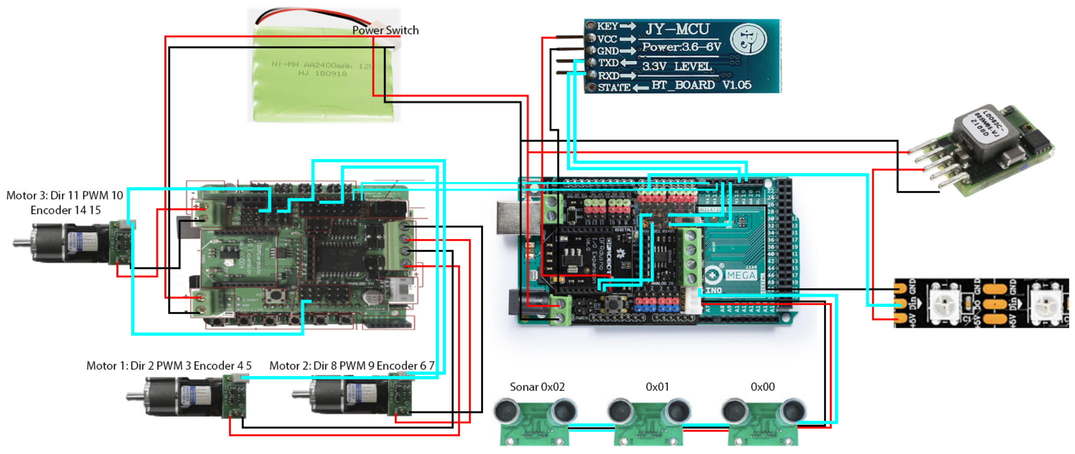


FIGURE 11: Flowchart showing the working principle of motor motion control

Besides the high-level connection plan of the inter board and electronic elements, a low-level plan on schematics level is provided in Fig. 12 for future PCB design of compact arduino expansion shield to further simplify and augment the system. The custom Arduino expansion consists of a MAX485 chip, a non-isolated DC-DC converter LDO03C, a circuit for generating inputs for L298 motor driver and exported pin connectors from Arduino Mega.

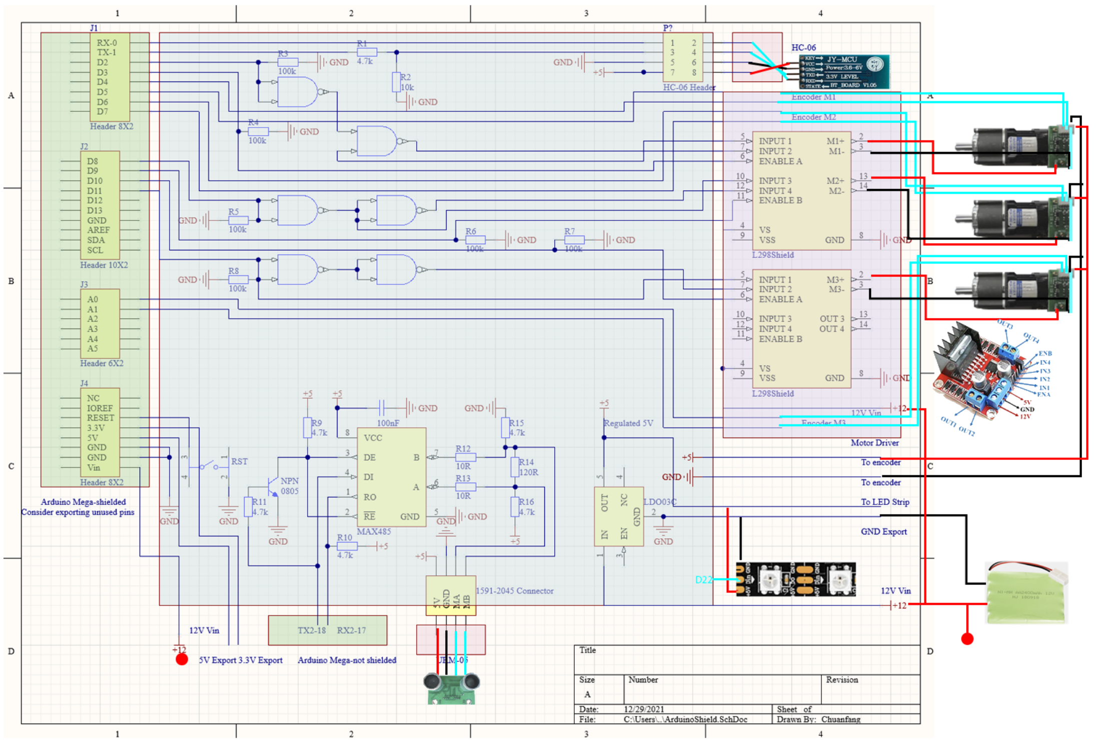


FIGURE 12: Customized Arduino shield design schematics

In the simplified system design, the AtMega-328 board and expansion would be removed and the Arduino Mega with a custom expansion shield and motor driver would accomplish all the task in the current design, which makes up a lite and extensive on-board system.

3.3 Embedded program logic

One of the main contents of this project is to enable the teleoperation of the mobile furniture with higher level language in ROS structure directly. This requires the basic functionality of sensors and actuators to be realized on-board. The embedded system behaves like a communication node for sending sensor data and a processing node for commanding actuators depending on ROS messages. This subsection describes details about the embedded program logic to realize this function.

3.3.1 Motion control

The working process of Omnibot motion control is explained as shown in the flowchart Fig. 13. Mega subscribes to the Twist type topic cmd_vel at 50Hz. It calculates the velocity and direction of 3 motors according to kinematics and encode it into an ASC-II encoded string of length 19 with format as defined in Fig. 14. The string contains 3 3-digit velocity ending with motor-identifier (left(l), right(r) and back(b)), 3 direction flags (0 for clockwise and 1 for counter-clockwise) ending with ’d’ and 1 end marker to help synchronize the serial communication. This message will be sent to the AtMega-328P with serial communication of 19200 baud rates, which can support the transfer of $\frac{19200\text{bits}}{8\text{bit}\times 50\text{Hz}} = 48$ characters at most. However, the transfer of string is not guaranteed to be synchronized at the start of first character and the AtMega-328p side will use the end marker star to decode the message to its original order.

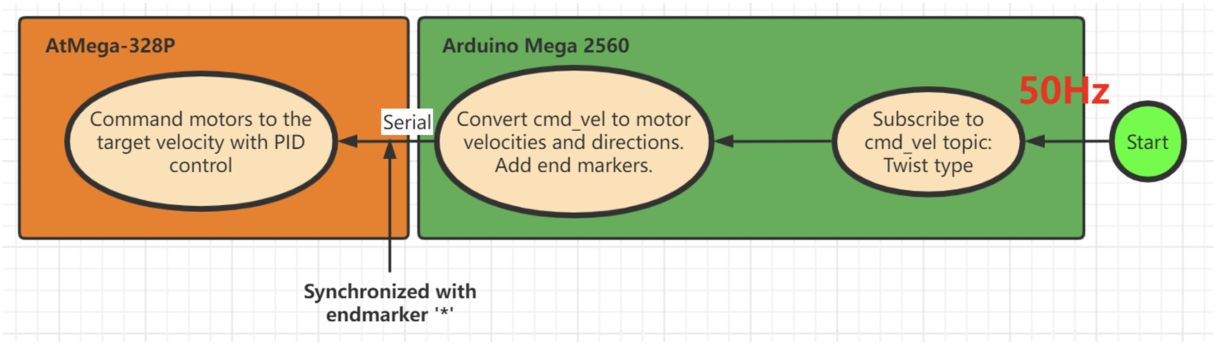

AtMega-328P
Command motors to the target velocity with PID control
Serial
Arduino Mega 2560
Convert cmd_vel to motor velocities and directions. Add end markers.
Subscribe to cmd_vel topic: Twist type
50Hz
Start
Synchronized with endmarker '*'

FIGURE 13: Flowchart showing the working principle of motor motion control

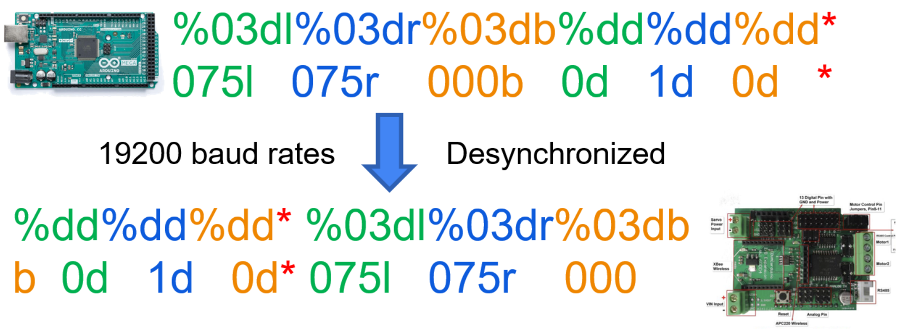

%03dl%03dr%03db%dd%dd%dd*
075l 075r 000b 0d 1d 0d *
19200 baud rates
Desynchronized
%dd%dd%dd* %03dl%03dr%03db
b 0d 1d 0d* 075l 075r 000

FIGURE 14: Defined format of motor command string

As shown in Fig. 15, the Twist velocity command (v_x, v_y, v_{yaw}) is converted to the motor velocities of 3 omni-directional wheels by solving the kinematics equation:

$$\begin{cases} \cos(\frac{\pi}{6}) \cdot v_{\text{left}} - \cos(\frac{\pi}{6}) \cdot v_{\text{right}} = v_{\text{x}} & (1) \\ \sin(\frac{\pi}{6}) \cdot v_{\text{left}} + \sin(\frac{\pi}{6}) \cdot v_{\text{right}} - v_{\text{back}} = v_y & (2) \\ (v_{\text{left}} + v_{\text{right}} + v_{\text{back}}) = \text{Radius} * v_{\text{yaw}} & (3) \end{cases}$$

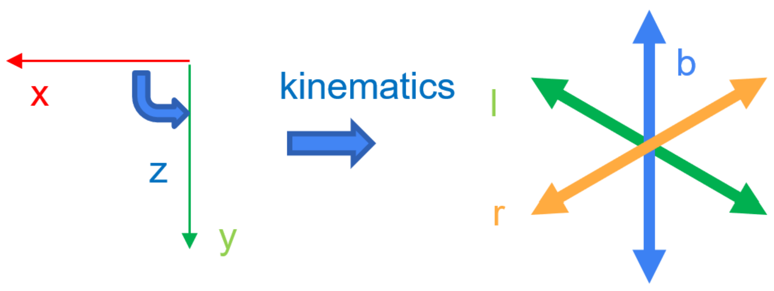


FIGURE 15: Defined format of motor command string

The computed motor velocities are multiplied by a compensation factor of 1.3 fed to the target output of the PID controller.

Fine tunning of the PID parameters requires to take into consideration the plant model in different situations (grounds with different viscous friction factor, Omnibot with different furniture load, etc.). Too many approximations have to be made and this would make the theoretically tuned optimal parameters useless in practice. For this project, the PID parameters are initialized to the recommended value from the Nexus demo program and tuned according to robot performance in practice. The final parameters of the PID are selected to be $K_p = 0.26$, $K_i = 0.02$, $K_d = 0.10$, which results in a smooth and reactive motion with the load of a chair on the floor of BioRob Laboratory.

Figs. 16–19 show the test results of Omnibot's performance in following different target velocities. The Omnibot is commanded to accelerate to the target velocity, keep moving at this velocity for some time and then brake. The data is collected using Optitrack and the missing data is handled with a moving-median filter with 1s window size. It can be seen that the PID controller is aggressive at high speed (0.30m/s, Fig. 19) and slow at low speed (0.15m/s, Fig. 16). But for the medium speed (0.20m/s and 0.25m/s) where Omnibot is configured to run, the performance is responsive and robust (Figs. 17–18).

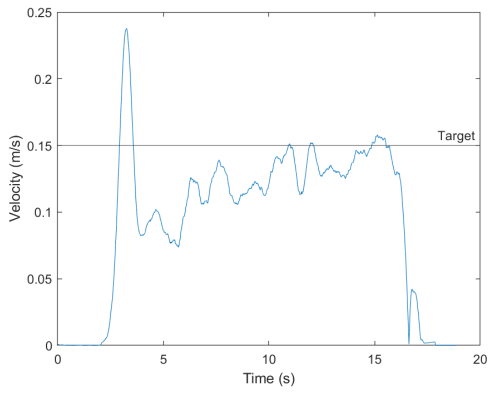


FIGURE 16: 0.15m/s target

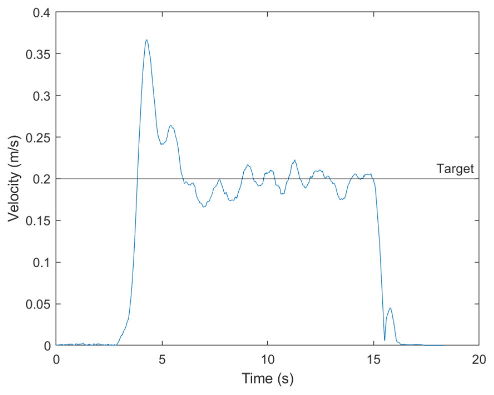


FIGURE 17: 0.20m/s target

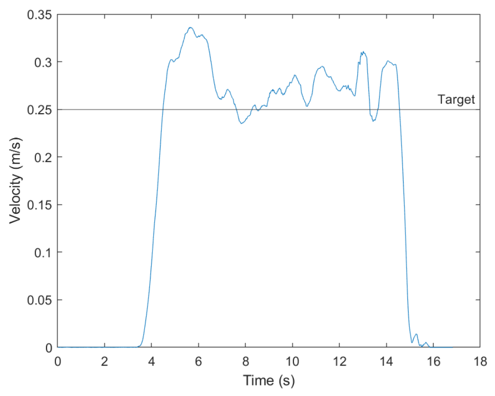


FIGURE 18: 0.25m/s target

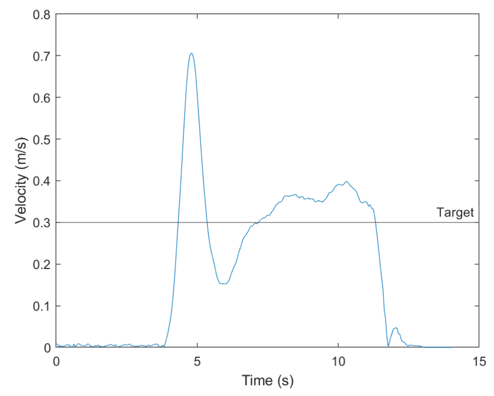


FIGURE 19: 0.30m/s target

3.3.2 Sonars

The pre-built platform comes with 3 URM-04 sonars and can be extended to up to 32 sonars in a chain. The Mega board communicates with sonars with message templates via RS-485 interface. Tab. 6 shows message templates used to configure and communicate with the sonar.

Command	Hex
Set device ids	0x55,0xaa,device,0x01,0x55,0xff,CHECKSUM
Trigger sonar measurements	0x55,0xaa,device,0x00,0x01,CHECKSUM
Read sonar measurements	0x55,0xaa,device,0x00,0x02,CHECKSUM
Read temperature	0x55,0xaa,device,0x00,0x03,CHECKSUM

TABLE 2: Message template for communicating with sonars

The "set device IDs" command is sent once at start up to initialize the sonars. The sonars are triggered one after another in order of the address. Each sonar requires 50ms to catch the reflected signal, convert it to distance and buffer it into the memory. Also, two sonars cannot trigger at the same time to avoid the interference between waves. Therefore, the sonar system is upper bounded to the frequency of $\frac{1000\text{ms}}{50\text{ms}\times 3\text{sonar}} \approx 6.(67)\text{Hz}$.

The URM04 sonar has a resolution of 1cm and a detection range of 4cm to 500cm. The mapping from distance to reading is shown as in Fig. 20.

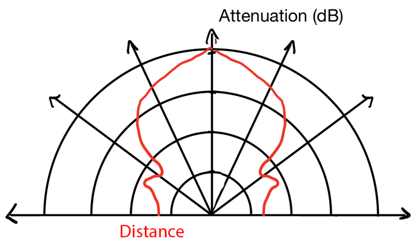


FIGURE 20: Detection range and readings of URM-04 sonar

It can be seen that with the same distance value from different directions (in red) the sonar returns different attenuations which correspond to different readings. This makes the local navigation based on sonars exclusively rather difficult. Also, considering the fact that the furniture legs could block the sonars from time to time, the sonar data is not used to implement local avoidance in this project. The sonar data is only examined at the ROS master for the availability of measurement and correctness of the framework as shown in Fig. 21. It could be interesting to add more sonars, better position sonars or combine sonars with other proximity sensors to explore the local avoidance in future projects.

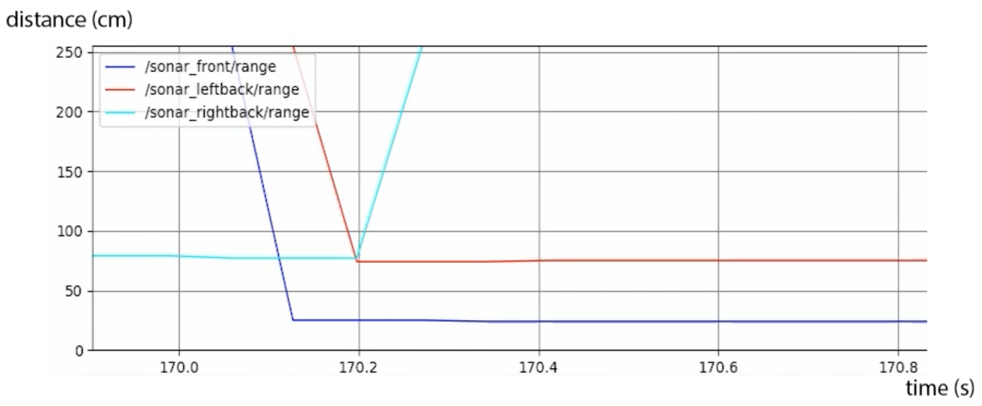


FIGURE 21: Sonar data streaming on ROS master

3.3.3 LED

An LED strip with 48 LEDs is pasted at the bottom of Omnibot and displays the status of motion. The LED strip consumes 2.4A current when fully lit up. This exceeds the limit current of Mega's regulator and an external non-isolated DC-DC converter LDO03 is introduced to power the LED strip directly.

The LED strip has 3 display modes in total.

- When the robot is still, 8 LEDs scattered on a hexagon light up in blue (0x0000FF).

- When the robot moves toward 1 direction, 9 LEDs on the forward direction light up. The middle LED is green (0x00FF00) and the side LEDs are yellow (0xFFFF00). Colors of LEDs between the middle one and the side one is gradient with Lightness Chroma Hue (LCH) method. Fig. 22.
- When the robot is turning, 8 LEDs scattered on a hexagon light up in red (0xFF0000). The LEDs rotate in the same direction of turning. Fig. 23.

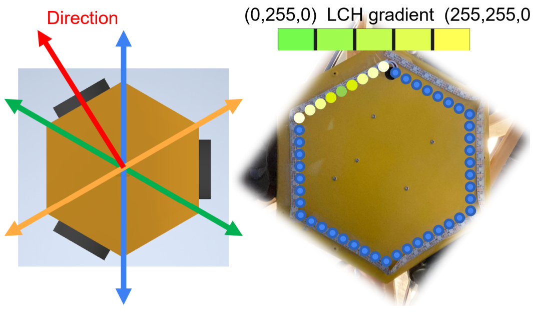


FIGURE 22: LED display moving mode

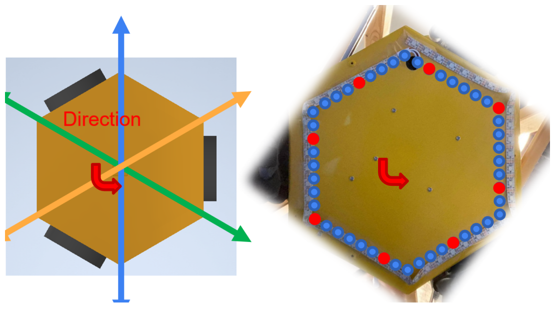


FIGURE 23: LED display turning mode

When the robot performs a motion combining moving and turning, the turning mode will dominant and be displayed.

3.3.4 Bluetooth

Omnibot is teleoperated via the Serial communication over Bluetooth JY-MCU at 115200 baud rate. 4 steps are performed to form the communication:

- The Bluetooth is configured with AT commands to run at 115200 bps.
- The Mega firmwire is configured with *ros_arduino* package to communicate with master at 115200 bps.
- The Bluetooth is paired with master PC as a serial resource with *rfcomm* commands
- The communication is built over the serial resource with *ros_serial* package given the baud rate defined above.

4 Algorithms

4.1 Mobile Furniture Localisation

The localisation of Omnibot in the baseline design is implemented with Optitrack (*Optitrack Documentation* n.d.), which localizes by tracking unique marker patterns. The markers are placed on the mobile furniture as shown in Fig. 24 and follows the following rules:

- Markers are placed in non-congruent unique patterns to distinguish bodies.
- A rigid body can be defined by at least 3 markers. But usually redundancy is needed to compensate for environment and material reflections.

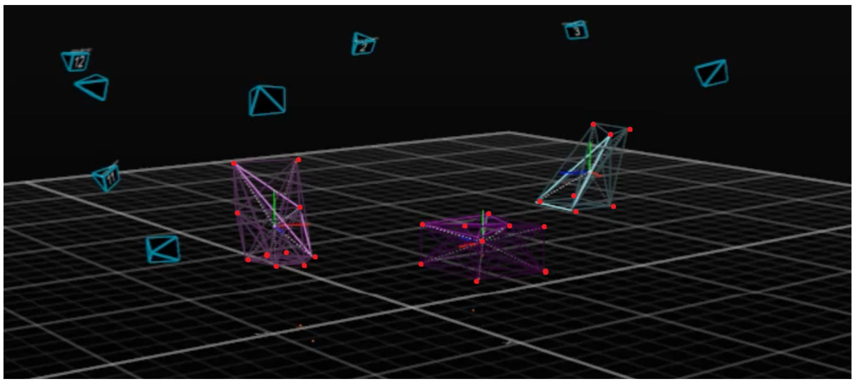


FIGURE 24: Marker placements on mobile furnitures

The detected rigid body poses are streamed to the ROS master via Virtual Reality Peripheral Network (VRPN)[1] at 120Hz. If the rigid body is temporarily blocked, the old pose data is used.

The pros and cons of this Optitrack localisation method are listed as follows:

+ Stable localisation against noise with redundant marker setup
+ Built-in motion data analysis pipeline for robots
+ High refresh rates with movie industry standards
– Requires careful device set up and calibration
– Localisation is limited inside the Optitrack arena

Although being a good method in testing and analysis, the Optitrack localisation is not suitable for real-world application due to its high requirements on hardware and environments. The localisation with the Optitrack is applied in this project to achieve the best testing performance of the robot in the design phase. In applications the method should be replaced with more general localisation methods based on vision/deep learning. The markers placed in a skeleton pattern on mobile furniture can also serve to validate results of another project on furniture skeleton localisation.

[1]http://wiki.ros.org/vrpn_client_ros

4.2 Mobile Furniture Navigation

The mobile furniture is expected to navigate around obstacles to go to desired positions. A simple test scenario is considered in which a user sitting on a chair commands the Omnibot to drive another chair to come in front of them, while avoiding a table in between. The baseline of the global navigation is implemented with a simplified version of visibility graph as described in Book Ben-Ari and Mondada 2017. Only 1 rectangular obstacle is considered in the simplified visibility graph. The path planning is done by checking the intersections between the path and the expanded obstacle.

The algorithm runs as shown in Alg. 1 and in Fig. 25.

Algorithm 1: Simplified Visibility Graph

```
1  while not reach the goal do
2      if intersection <= 1 then
3          go straight to the target;
4      if intersection = 2 then
5          if intersection on adjacent edges then
6              go to corner node between the intersection;
7              go straight to target;
8          else
9              go to closest node(NODE 2) to the target via
10             the closer node(NODE 1) of the two closest nodes to robot to NODE 2;
11             go straight to target;
12         end
13     if intersection = 3 then
14         Robot aligned with obstacle edge
15         go straight to target;
16     if intersection = 4 then
17         Intersection across diagonal
18         go to one of either side of diagonal;
19         go straight to target;
20 end
21 while not align with the goal do
22     rotate to align with the goal;
23 end
```

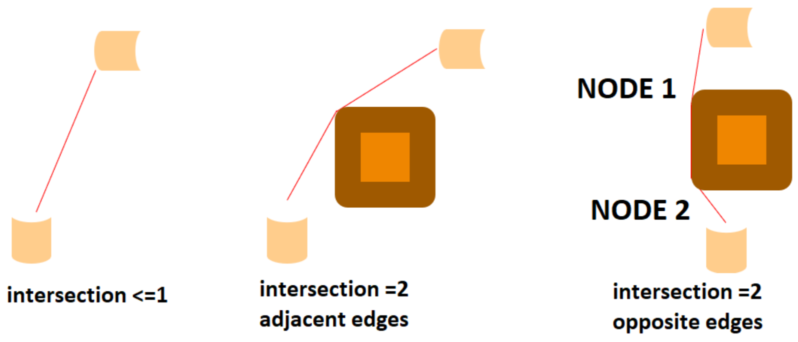
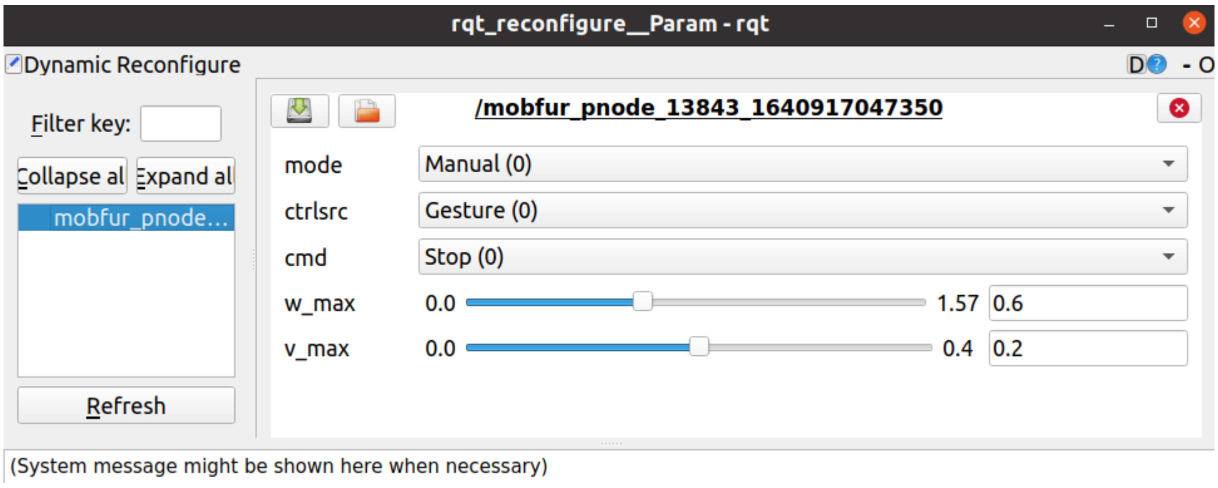


FIGURE 25: Robot path with simplified visibility graph

To avoid oscillations at obstacle nodes and avoid robot from entering the obstacle due to localisation error, the intersection is calculated on a contracted version of obstacle boundary. To avoid oscillation at target and alignment, the robot performs a slow stop and the motion is programmed with a dead-zone in which command only activates when the error is greater than a threshold ($5\mathrm{cm}$ or $5°$).

4.3 Interactive user Interface

The Omnibot can be controlled with an interactive user interface built with the ROS package called *dynamic_reconfigurer*[2] and shown in Fig. 26.



/mobfur_pnode_13843_1640917047350

mode	Manual (0)
ctrlsrc	Gesture (0)
cmd	Stop (0)
w_max	0.0 – 1.57: 0.6
v_max	0.0 – 0.4: 0.2

(System message might be shown here when necessary)

FIGURE 26: Interactive user interface

The parameters that can be changed in the interface and corresponding effects are presented in Tab. 3

[2]http://wiki.ros.org/rqt_reconfigure

Parameters	Values	Effects
mode	Manual	Enter manual control mode
	Automatic	Enter global navigation
	Debug	Stop and print verbose debug info
ctrlsrc	Voice	Voice command (Subsec. 4.4) takes over manual control
	Gesture	Gesture command (Subsec. 4.5) takes over manual control
	Tablet	Tablet command (Subsec. 4.6) takes over manual control
cmd	Stop	Stop if no other control sources are activated
	Forward	Move forward if no other control sources are activated
	Backward	Move backward if no other control sources are activated
	Left	Move left if no other control sources are activated
	Right	Move right if no other control sources are activated
w_{max}	0.0 to $1.57 radians/s$	robot turning velocity
v_{max}	0.0 to $0.4 m/s$	robot moving velocity

TABLE 3: Interactive user interface parameters

This interface is the top-level control interface for Omnibot, which not only controls the robot directly, but also switches between different interactive modes. People with limited mobility can suffer from other issues impacting on their interaction abilities as well. Typical disorders are above-elbow amputee or the extremities paralysis by the spine cord injury. The intelligent assistive robot design often comes with various control methods to help address this problem (Moon et al. 2003). In this project, a total of 4 different kinds of interactive controls are implemented to help people with different needs or different preferences.

- Program interface designed for carer of the people for configuration purposes.
- Voice control designed for people who cannot move their upper body.
- Mobile application control designed for people who can move their upper limbs and wants to control the robot in an intuitive way.
- Gesture control also designed for people who can move upper-limbs but not with enough precision for operating tablets.

4.4 Voice control

Omnibot can be controlled with voice commands in the interactive mode. The voice control is implemented with *DeepSpeech*[3], an open-source speech-to-text engine (Hannun et al. 2014, Amodei et al. 2015). The core of DeepSpeech framework is a recurrent neural network (RNN) with 5 hidden layers which ingests speech spectrograms and generates English text transcriptions. The network structure is shown in Fig. 27.

[3]https://deepspeech.readthedocs.io/en/r0.9/

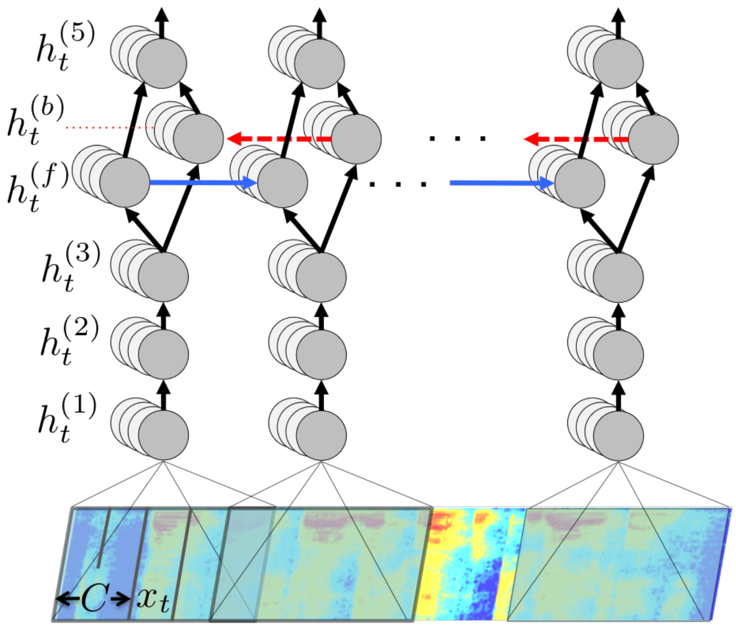


FIGURE 27: Structure of DeepSpeech RNN model and notation

In the project, a real-time voice transcription script is implemented with DeepSpeech framework as shown in Fig. 28.

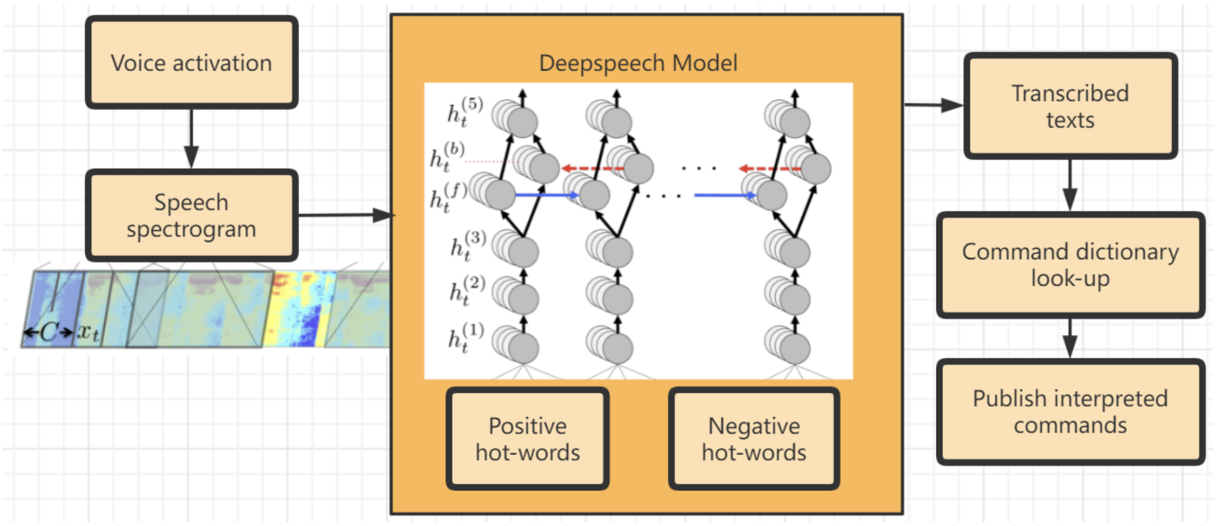


FIGURE 28: Flowchart of voice commands recognition

The script takes in a speech spectrogram truncated by the voice activation feature. The spectrogram goes through pre-trained DeepSpeech model with a hot-words dictionary. In the dictionary the desired commands are boosted and the words that sound close to commands are penalized (eg. ”left” is boosted and ”let” is penalized). A look-up is performed on the transcribed texts in the predefined command list in Tab. 4. Once matched, the commands would be published to the Omnibot motion control node.

Command	Effect
Automatic	Enter global navigation mode if not already; Do global navigation as in Subsec. 4.2
Stop	Enter manual mode if not already; Stop
Forward	Go forward if in manual mode
Backward	Go backward if in manual mode
Left	Go left if in manual mode
Right	Go right if in manual mode

TABLE 4: Pre-defined command list

According to the table, voice commands can control the Omnibot to do the global navigation as introduced in Subsec. 4.2 or manually. In the manual mode voice commands set a target $5\mathrm{m}$ in front of the commanded direction and navigates the robot towards it. During the run the robot direction is continuously calibrated with the localisation information.

Although the DeepSpeech model is rendered more stable against pronunciations and accents thanks to the hot-words dictionary, misinterpretations still occur in noisy environments. A command is usually better recognized in a context than stand-alone. Also, the voice activation feature could lead to loss of information. Due to 2 points above, it is recommended to utter the critical words (command) at 2nd place or later of a sentence within meaningful context. (Eg. The sentence "Robot please go forward" has a higher chance of being correctly interpreted than shouting out "FORWARD!")

Also, since the nature of DeepSpeech is a universal voice-text engine. The output of the network includes all possible texts other than the commands. This results in a lower performance and lower accuracy in interpretation. For this specific task, it could be a better idea to train a simple multi-output network for a simple task of classifying commands.

However, the universal nature of DeepSpeech also brings convenience. The command list could be easily expanded or edited by changing the hot-words dictionary without retraining the model and regenerating the dataset, which makes the baseline design quite extensive in the future. This is the main reason why DeepSpeech framework is used in this project.

4.5 Gesture control

Omnibot can be controlled with gesture commands in the interactive mode. The gesture control is implemented with *MediaPipe framework*[4] (Lugaresi et al. 2019), which offers a detection method for Hand Landmark Model as shown in Fig. 29.

[4] https://google.github.io/mediapipe/

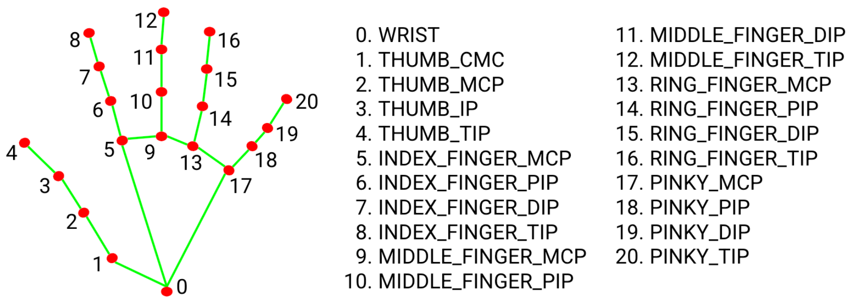


FIGURE 29: Hand Landmark Model

The detection of Hand Landmark Model consists of 2 steps as shown in Fig. 30. First, the palm is detected over the whole image with Single Shot MultiBox Detector (SSD) (W. Liu et al. 2015). Then, the 21 landmarks are obtained via regression on the palm pixels.

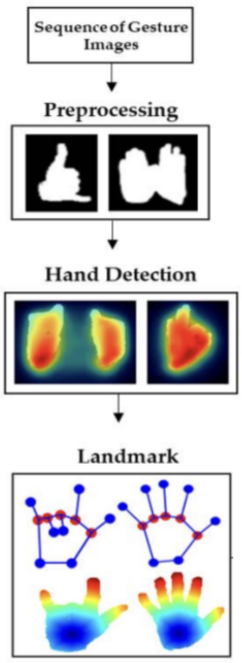


FIGURE 30: Hand Landmark Model detection pipeline.

In this project, only the most significant palm is detected to make the Landmark Model. And a further step is performed to predict the gesture of the hand:

- Curved: Detecting a finger is curved or not by comparing the vector direction of $\overrightarrow{(MCP, PIP)}$ and $\overrightarrow{(PIP, TIP)}$.
- Direction: Detecting the finger direction by doing arctan2 of vector $\overrightarrow{(MCP, PIP)}$.

The gestures can be classified in to 6 commands as shown in Tab. 5 with the help of 2 information above.

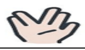
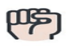
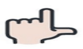
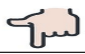
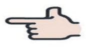
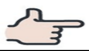
Gesture	Curved	Direction	Effect
	None	-	Enter automatic mode
	All	-	Enter manual mode and stop
	Not index finger	Up	Move forward if in manual mode
	Not index finger	Down	Move backward if in manual mode
	Not index finger	Left	Move left if in manual mode
	Not index finger	Right	Move right if in manual mode

TABLE 5: Gesture command list

The command operations in gesture control mode are exactly the same as in voice control mode. Also, it is observed that the gesture control is much more responsive (over 20Hz with CPU) than voice control, as the multi-box detection is a binary pixel classification in nature and is much lighter than the voice transcription which has thousands of text outputs as classes.

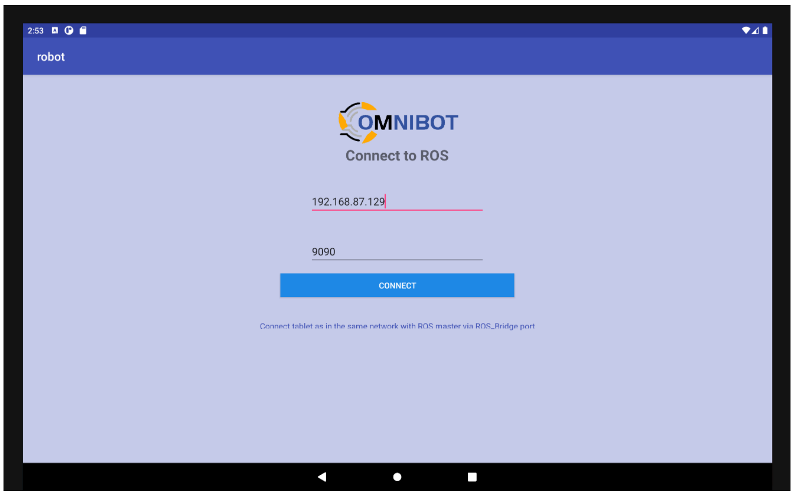
Similar to DeepSpeech, the gesture lists could be easily extended by defining new relations between different landmarks without re-training any models, which makes the project easily extendable in the future.

4.6 Tablet control

Omnibot can be controlled with a tablet application on Android in the interactive mode. The application is developed with Java in Android Studio.

The application communicates with ROS master over TCP/IP via *ros_bridge server*[5]. For connection the mobile device has to be in the same network as the master PC and the IP address and server port of ROS master is required. The welcome screen is shown in Fig. 31

[5] http://wiki.ros.org/rosbridge_server



2:53
robot
OMNIBOT
Connect to ROS
192.168.87.129
9090
CONNECT
Connect tablet as in the same network with ROS master via ROS_Bridge port

FIGURE 31: Welcome screen of the tablet application

Once the connection is formed, the application pops out a message and goes into main activity as shown in Fig. 32. The Omnibot control is performed in this main activity page. There are 2 control modes in the main activity page. In the automatic mode, the robot would perform the global navigation as described in Subsec. 4.2, and all other elements on the screen are locked. In the manual control mode, other elements are unlocked and can be accessed for motion control. The joystick controls the direction and velocity of the robot. The left and right buttons control the turning of the robot. The robot could move and turn at the same time. There is a safety measure to prevent the left and right arrows being pressed at the same time.

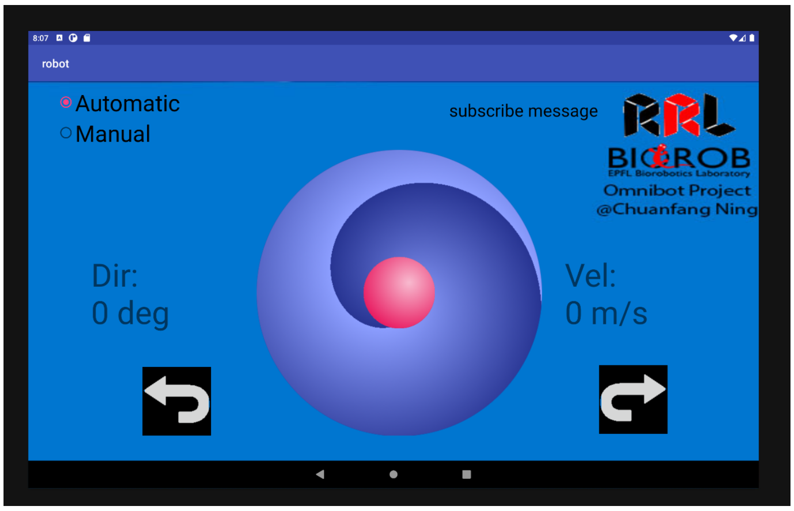


FIGURE 32: Main activity of tablet application

Besides sending commands to ROS master, the main activity page could also subscribe to topics and show the messages from the master. This function could be extended to allow the user to have a full monitor of robot status remotely.

5 Conclusion

By the end of the project, a full baseline for adapting a pre-built robot platform to Omnibot is successfully designed, implemented and verified.

The mechanical baseline allows the Omnibot to drive the provided furniture smoothly and firmly at medium velocity. The design is easy and economical to replicate. The further extension of the baseline with extra layer is analyzed and discussed.

The electronic baseline allows the teleoperation of motors, sonars and LEDs on the Omnibot via Bluetooth. The new system is abundant with resources for further extension. A more compact solution is raised to integrate the current system and reduce redundancies.

The algorithm baseline realizes the basic functionalities of Omnibot as a mobile robot. The localisation is implemented with a friendly style to combine with the furniture skeleton localisation project by Lixuan. The navigation accomplishes the basic task of obstacle avoidance in a simple setup. The interactive control (user interface, voice, gesture, tablet) provides various control styles that could fit different users. Above all, the baseline algorithms are implemented in a way that is easily extensible for future projects.

The Omnibot follows the traditional style of mobile robot and goes a plainer way in mobilizing furniture than another famous robot Roombot (Hauser et al. 2020) in the laboratory. At the cost of modularity and locomotion varieties, Omnibot has increased mechanical strength, better extensibility and lower construction cost. With proper extensions in future projects, Omnibot is potential to commercialize and benefit people with unlimited possibilities.

6 Appendix

Demo videos are shared with EPFL-wide Google Drive at

https://drive.google.com/drive/folders/1gyu45ZkrJY6R2dZy_43KalAXH-d8TCW6?usp=sharing

Source codes, configs and setup guides are documented at internal Git Repository at

https://ponyo.epfl.ch/students/mobfur

with the following structure:

```
├── Mechanics  # Folder for mechanics part
│   ├── BOM_Lists  # Bill of Materials in extending one Omnibot
│   ├── Connection # Illustration of mechanical attachment configuration
│   ├── Models  # Model files of extended robot platform created in Inventor
├── Electronics   # Folder for electronics part
│   ├── Arduino_codes  # Codes for sensing/actuating/teleoperating the system on board
│   ├── Libs  # Library dependencies for building embedded C codes
│   ├── Wiring  # Wire plan of the embedded system and design for the new expansion shield
│   ├── Schematics  # Useful schematics
├── Mobfir_basics_ws  # Workspace with source files for algorithm part
│   ├── src
│   ├──    ├── mobfur_navi  # main package for Omnibot motion and navigation
│   ├──    ├── optitrack_streaming  # package for Optitrack localisation
│   ├──    ├── voice_ctrl_streaming  # package for streaming voice commands
│   ├──    ├── gesture_ctrl_streaming  # package for streaming gesture commands
├── Android UI # Android Studio project for interactive tablet control
├── Optitrack # Optitrack configuration and marker models
├── img # Image folder for README
└── README.md
```

FIGURE 33: Repository structure

The project timeline is shown as in Tab. 6.

Week	Work
1	Draft for mechanical connection, modelling of parts and attachments
2	Discussion and refinement for mechanical design,improve models, place orders
3	ROS control structure setup, wired control of available sensors/actuators
4	Wireless control of sensors/actuators via Bluetooth
5	Orders arrive, mechanical assembly and engineering
6	Motion control and localisation with Optitrack
7	Global navigation with simplified visibility graph
8	Mid-term presentation, document review for interactive controls
9	Interactive voice control
10	Interactive gesture control
11	User interface for mode switch and manual commands
12	Android App development
13	Android App development
14	Integral test, code clean and project wrap up

TABLE 6: Timeline of the project

References

Yanco, HA and John Aronis Richard Simpson (1998). *Assistive Technology and Artificial Intelligence*.

Zisis, Eleftherios et al. (Aug. 2021). 'Digital Reconstruction of the Neuro-Glia-Vascular Architecture'. In: *Cerebral Cortex* 31.12, pp. 5686–5703. ISSN: 1047-3211. DOI: 10.1093/cercor/bhab254. eprint: https://academic.oup.com/cercor/article-pdf/31/12/5686/40814577/bhab254.pdf. URL: https://doi.org/10.1093/cercor/bhab254.

Vouga, Tristan et al. (2017). 'TWIICE—A lightweight lower-limb exoskeleton for complete paraplegics'. In: *2017 International Conference on Rehabilitation Robotics (ICORR)*. IEEE, pp. 1639–1645.

Liu, Dong et al. (2018). 'EEG-based lower-limb movement onset decoding: Continuous classification and asynchronous detection'. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26.8, pp. 1626–1635.

Optitrack Documentation (n.d.). https://v22.wiki.optitrack.com/index.php. Accessed: 2022-01-05.

Ben-Ari, Mordechai and Francesco Mondada (2017). *Elements of robotics*. Springer Nature.

Moon, Inhyuk et al. (2003). 'Intelligent robotic wheelchair with EMG-, gesture-, and voice-based interfaces'. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 4. IEEE, pp. 3453–3458.

Hannun, Awni Y. et al. (2014). 'Deep Speech: Scaling up end-to-end speech recognition'. In: *CoRR* abs/1412.5567. arXiv: 1412.5567. URL: http://arxiv.org/abs/1412.5567.

Amodei, Dario et al. (2015). 'Deep Speech 2: End-to-End Speech Recognition in English and Mandarin'. In: *CoRR* abs/1512.02595. arXiv: 1512.02595. URL: http://arxiv.org/abs/1512.02595.

Lugaresi, Camillo et al. (2019). 'MediaPipe: A Framework for Building Perception Pipelines'. In: *CoRR* abs/1906.08172. arXiv: 1906.08172. URL: http://arxiv.org/abs/1906.08172.

Liu, Wei et al. (2015). 'SSD: Single Shot MultiBox Detector'. In: *CoRR* abs/1512.02325. arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

Hauser, S. et al. (2020). 'Roombots extended: Challenges in the next generation of self-reconfigurable modular robots and their application in adaptive and assistive furniture'. In: *Robotics and Autonomous Systems* 127, p. 103467. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2020.103467. URL: https://www.sciencedirect.com/science/article/pii/S0921889019303379.