Romi A.B.C

Group Members:
Carol Yan, Bo Pang, Azamat Siddiqui

Objective

With the pandemic rapidly spreading and posing challenges for in-person working environments, we set out to explore the potential of remote controlling of complex systems. The project aims to implement a signal transmission pipeline and control system that supports remote control via BLE and networking.

In addition, the complexity of current-day technologies is fast increasing, and thus may require cooperation between several control clients. Our project also supports simultaneous incorporation of multiple client inputs to control one single robot, with each client responsible for a specific set of functions in the overall complex system. Such a functionality opens the way for many complicated controlling machineries such as remote vehicles, medical surgery, and construction equipment.

System Design Overview

Hardware

Our robot is mostly based on the “Romi” platform built during labs, with the addition of an arm assembly, 3 servo motors that take 5V DC and PWM inputs, and a mount for mobile devices to stream video. The arm is capable of three actions: lift (raise the arm up or down); tilt (pitch the gripper up or down) and grab.

The “Arm Kit for Romi” made by Pololu [1] provides enough parts to mount the arm assembly neatly on our Romi platform. Mounting the buckler, nRF52 board, and a phone holder on top of that requires some creativity but is not too difficult. Our Romi set up is shown in *Figure 1* and *Figure 2* below.

System Architecture

At a high level, our system consists of multiple (3 for us) “hand-held controllers” and a robot. Each hand-held controller samples hand movements, calculates the corresponding robot actions, and broadcasts the command via BLE. The robot receives commands from multiple controllers (via BLE scan), merges them, and executes the commands.

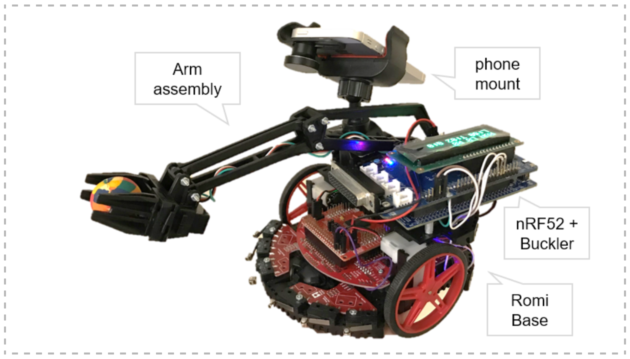


Figure 1: Structure of Romi A.B.C

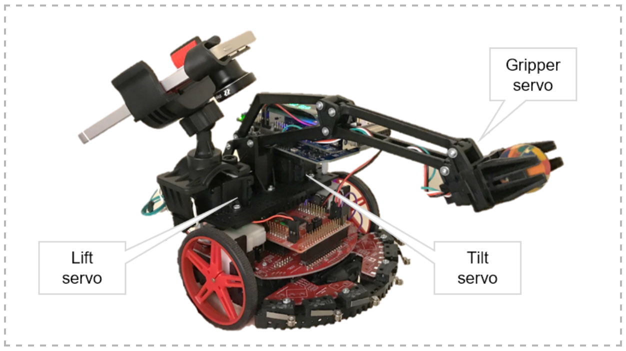


Figure 2: Structure of Romi A.B.C

To make collaborative remote control possible, we also use computers (with BLE dongles) to relay the BLE commands over the Internet. Two of us control the robot remotely, so we set up two local computers doing BLE broadcasting on behalf of them.

With-in each subsystem, the setup is similar to what we used in labs; with the addition of BLE capabilities, and PWM drivers. *Figure 3 (on the next page)* illustrates the system architecture with more details.

Software Design Overview

Hand-held controller signal mapping

We obtained pitch and roll angle signals from the digital accelerometer on each hand-held controller. We used the switch signal to determine which part of Romi movement the angle data is responsible for. (e.g. it controls arm lifting if the switch is set to low, and arm tilting and grabbing if otherwise.)

To achieve smooth control, we average the latest 3 signals sampled, and map it to the robot’s movement (speed w/ direction) using affine functions shown in *Figure 4 (on the next page).*

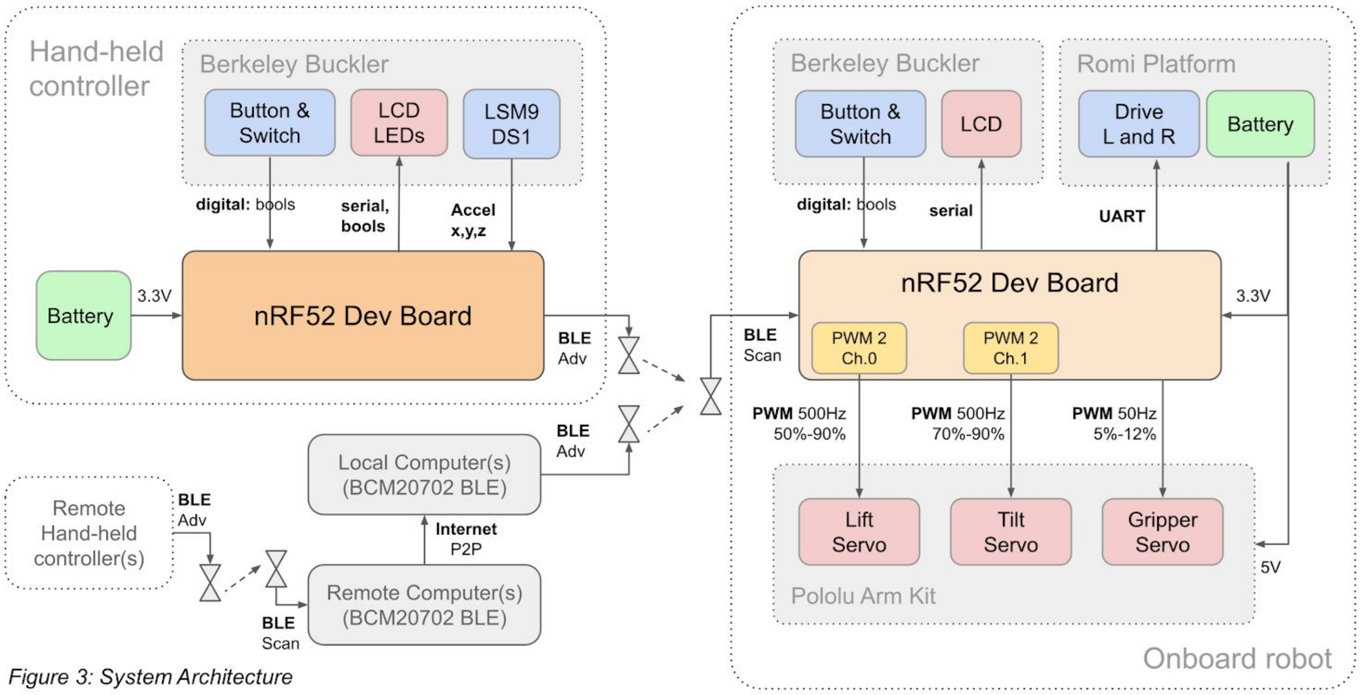


Figure 3: System Architecture

We then encode the movement commands into a payload array to be broadcasted over BLE. Payloads from multiple controllers are merged by the robot, as shown below in *Figure 5.*

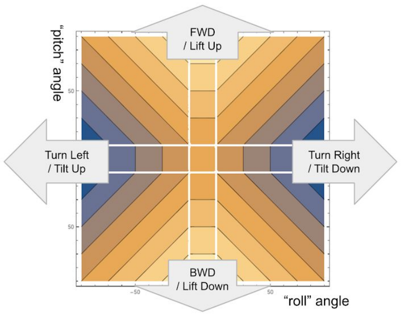
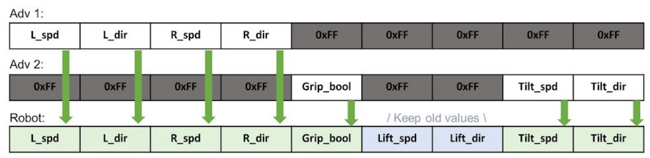


Figure 4: Plot of mapping function (contour: speed of movement)



Adv 1:	L_spd	L_dir	R_spd	R_dir	0xFF	0xFF	0xFF	0xFF	0xFF
Adv 2:	0xFF	0xFF	0xFF	0xFF	Grip_bool	0xFF	0xFF	Tilt_spd	Tilt_dir
Robot:	L_spd	L_dir	R_spd	R_dir	Grip_bool	Lift_spd	Lift_dir	Tilt_spd	Tilt_dir

/ Keep old values \

Figure 5: Payload encoding & Merge methods

Controller-machine BLE communication

We utilized Python subprocess module [3] and bleak library [4] to advertise BLE signals to the robot (local side), or fetch advertisements from hand-held controllers (remote side).

Server-client P2P networking

In order to enable remote control of the Romi, we applied Peer-to-Peer network concepts and set up clients and server nodes. We utilized Python socket library [2] and selected to use the TCP protocol. For simplicity, we port-forwarded the public IP address of the machine that set up the server socket so that the client socket can reach the server directly and efficiently. After the connection is established, the server socket keeps listening for input every 100 milliseconds and the remote-side client sends payload signals at the same rate (or as fast as the BLE dongles can scan).

Arm control & PWM

The Romi's arm is controlled by 3 servo motors via PWM signals. The Lift & Tilt motors need persistent torque (PWM inputs) to keep arm position, so we used dedicated PWM generators (based on timer-2 and comparator interrupts) for them. The gripper has a strong gear reduction so its servo does not have to generate persistent torque. Every time the gripper needs to move, we just use a software loop to generate a short pulse (160ms) of PWM for it.

Initially, we mapped the pitch and roll angle signals to the speed of arm movements. However, upon testing on remote setup, we realized it's difficult to let the arm stay at desirable positions due to latency. To resolve this issue, we changed the design to map the signals to the absolute positions of the arm so that we can hold the hand-held controller until the arm responds and adjust accordingly. ***This video*** illustrates this change.

Figure 6 (on the next page) gives an overview of our software design.

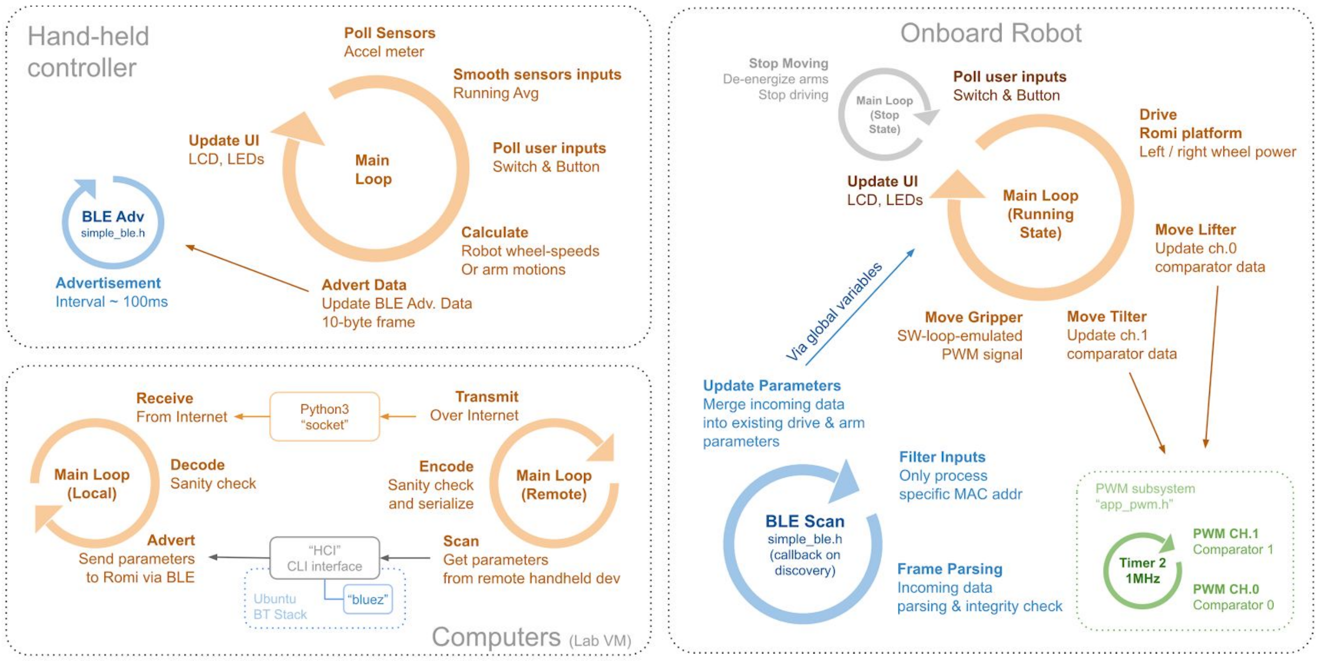


Figure 6: Software Architecture

Connection to Course Topics

Hierarchical State Machines

The project started off with just a simple state machine that showed how the robot wheels moved on the controller side. After integration of the robot arm, a hierarchical state machine was created that had two high-level states; Drive and Arm. Both of these states had other states embedded in them.

Concurrent State Machines

On the robot side, we had a concurrent state machine; it can act on both the state machines mentioned above if there are more than one controller. For example, if one controller was in the Drive state and another controller was in Arm state, the robot would act on both the commands received simultaneously.

Interrupts and timers:

The main loop consistently gets interrupted by the BLE advertisements. If the advertisements are from a controller (MAC filtering), then the data received will be articulated to the new actions that the state machine should take. The above is how the concurrent machines are made possible in the context of this project. The project needed to provide different PWM signals for the servos embedded in the robot arm, so timers were used to create different PWM waveforms.

Discrete data processing:

The gesture data read from the digital accelerometer is discrete. To smoothen the control of the robot, it is filtered by a three-point moving average and rounded to the nearest integer. Additionally, the data is thresholded before advertising to avoid driving the hardware to limit.

System Modelling and Design:

The raw data taken from the digital accelerometer were modeled as pitch and roll angles to act as gesture control signals. Later by incorporating the switch input on the Buckler the same measurements were modeled to act as control signals for the arm control (or many other combinations).

Execution time analysis of code Stack:

Because of the nature of this project there had to be done some practical measurements and analysis of code stack. Particularly, the time delay for the BLE “scan” operation on our remote computer, running Ubuntu and its “bluez” BT stack.

System Evaluation & Results

Arm control & Delay assessment

2 of us control the arm remotely. Below is the typical delays we see (in milliseconds):

	Typical	Worst-case
Internet	80	150
Remote BLE (scan)	500	1000
Local BLE (advertisement)	100	250

Table 1: Delay breakdown

As shown by the table above, the majority of the delay comes from the BLE dongle on remote computers, which cannot scan as fast as the nRF52’s BLE driver does. We put a lot of effort into this issue, but this is the best we can do for now. We suspect it’s related to the “bluez” BLE driver stack on Ubuntu Linux.

Thanks to our improved arm controlling method (mentioned above on system-overview/software), we can remotely control the arm just fine.

Drive control
Driving the robot is more sensitive to delays than arm control, so it is done locally. Taking advantage of our affine gesture-to-drive-speed mapping, we can either drive our robot very slowly - to precisely position it in front of an object - or very fast to cover distances. Changes in speeds are done very smoothly. ***This video*** illustrates our controllability on robot driving.

Flexibility of controller configuration
As mentioned above, the robot scans for multiple controllers and merges their inputs. We tested varieties of controller configurations (such as “drive / lift / tilt & grip”; versus “drive & tilt & gripper / lift”; versus “all-in-one-controller”), the robot always responded as we expected.

The design is not perfect, though. As we use more controllers to control a robot, the robot responds noticeably slower. That’s not surprising, given the robot needs to process more BLE advertisements in that case.

Hardware reliability
Last but not least, the hardware. Our robot’s structure holds up very well after many hours of moving & driving. The rear mounted arm system helps balance the robot when grabbing objects (it can lift 200g).

In terms of power supply, our servo motors are pretty efficient upon our testing (amp under 5VDC):

	Lifter	Tilter	Gripper
worse-case current	1.5A	1.5A	0.8A

Table 2: Servo motors power requirements

The power distribution board comes with the Romi platform (rated 5V 7A) handles the workload well. No parts noticeably heats up after a long operation.

Conclusion

The project was designed to be completed in three stages with the following goals in mind:

1. Create a simple state machine to control the robot and establish bluetooth data transmission between gesture controller and NRF-52 board on the robot.

2. Integrate the Pololu arm on the robot by modifying the hardware and make sure the power distribution is not a cause of failure. Extend the controller state machine to send control signals to control the Pololu arm as well. On the robot side add the capability to provide those specific signals to the arm.

3. Create a peer to peer connection over the internet to control the robot remotely. A controller would send BLE data to the client computer which in turn would forward that to the host server. The server would broadcast the control signals to the robot as BLE data so the robot can capture it.

All of the goals set at the beginning of the project were achieved. Additionally, the capability of multiple remote controllers were added so that a robot could be controlled by multiple controllers remotely at the same time.

References

1. “Pololu - User's Guide for the Robot Arm Kit for Romi.” *Pololu Robotics & Electronics*, www.pololu.com/docs/0J76/all.

2. “Socket - Low-Level Networking Interface.” *Socket - Low-Level Networking Interface - Python 3.9.1 Documentation*, docs.python.org/3/library/socket.html.

3. “Subprocess - Subprocess Management.” *Subprocess - Subprocess Management - Python 3.9.1 Documentation*, docs.python.org/3/library/subprocess.html.

4. Blidh, Henrik. “Bleak.” *PyPI*, pypi.org/project/bleak/.

5. Project repo, https://github.com/CarolNiuYan/149Projec