UR5 Plays JENGA

Chia-Hung Lin, *Robotics MSE Class of 2019, JHU*

***Abstract*—In this project, the author attempts to make a Universal Robot UR5 robot arm to play Jenga autonomously using Robot Operating System as the main framework and designed Jenga specific end-effector that includes a gripper, a force probe, and a range finder. The system is fully autonomous and can play a block in around 60 seconds. The highest record of the level reached by the robot is 21.**

I. Project Goals

A. Need

Robotics Jenga players often not play under official rules and base on proprietary software and hardware architecture.

B. Goal

A robot agent that plays Jenga on itself or against human players.

C. Objectives

- System is fully autonomous; no human intervention required.
- When playing on itself, reach at least 20 level before the tower collapses. That is, extract at least six blocks from the tower.
- A 60 second time limit for the robot to extract a block and put it on top of the tower.

D. Constraints

- The implementation is on the UR5 robot arm
- The implementation uses ROS.
- The workspace is the table area in Wyman 170.
- Robot plays on full size, 18-level tower.

II. System Design

A. Functional Structure

Typical human strategy for playing Jenga involves four major moves:

1) Look at the tower and decide which block to extract.
2) Feel the looseness of the block and push it in some distance.
3) Pull out the block on the other side.
4) Put the block back on top of the tower.

Therefore, in observation of human strategy, the system decomposes into the following functional blocks.

1) Locate the tower
2) Sense looseness of blocks
3) Localize the salient block
4) Extract and return the block

This project is inspired by demonstrations of a robot playing Jenga by Torsten Kroger [1] [2] in 2008. JENGA® is a registered trademark owned by Pokonobe Associates.

B. Systems Diagram

Please refer to Figure 1 for the implemented systems diagram. A ROS core is running on a workstation PC, with AI player, trajectory generation, AR marker detection, waypoint broadcaster, and end-effector firmware being separate ROS nodes, exchanging messages via ROS core.

III. Infrastructure and Hardware

A. Infrastructure

- **UR5:** one of the Universal Robots UR5 robot arm in Wyman 170 is utilized for this project. The UR5 is a 6 degree-of-freedom collaborative robot arm that has a 850mm radius and 5kg payload capacity.
- **Jenga:** a classic Jenga tower consisting of 54 hardwood blocks that are roughly $15 \times 25 \times 75\text{mm}$ in size. Every block is subtly different in dimensions so as to create a stochastic stacking process that make the tower vary from game to game.
- **Logitech C920 HD USB Webcam:** one usb camera is used in the project for locating the AR markers. The camera is attached to a tripod that is placed on the table, looking down on the tabletop.

B. Hardware and Electronics Design

In order to let the UR5 play Jenga, I have designed an end-effector from scratch to emulate human strategy. The end-effector is designed to be as compact as possible to maximally utilize the workspace. It is about $88 \times 85 \times 80\text{mm}$ in size with the electronics and without the probe on the load cell. A 55mm wide AR marker is attached to the back of the end-effector for hand-eye calibration. Please refer to Figure 2a for the solid model, and Figure 3 for a photo of the assembled end-effector on the robot. Some key design points are as followed:

- **Locate the tower:** AR markers
  - Originally, the system would have a computer vision system that analyze the tower composition dynamically. However, due to time limitations, a localization system heavily relying on AR markers is implemented.
  - Four 90mm wide square AR markers are printed on a paper, and each marker is separated by 185mm. This distance is specifically calculated to leave a 10mm white space around the markers, and space to put the 75mm wide Jenga tower in the middle of the markers.
  - Please see the software design section for implementation details on inferring the pose of the Jenga tower from one or more markers.

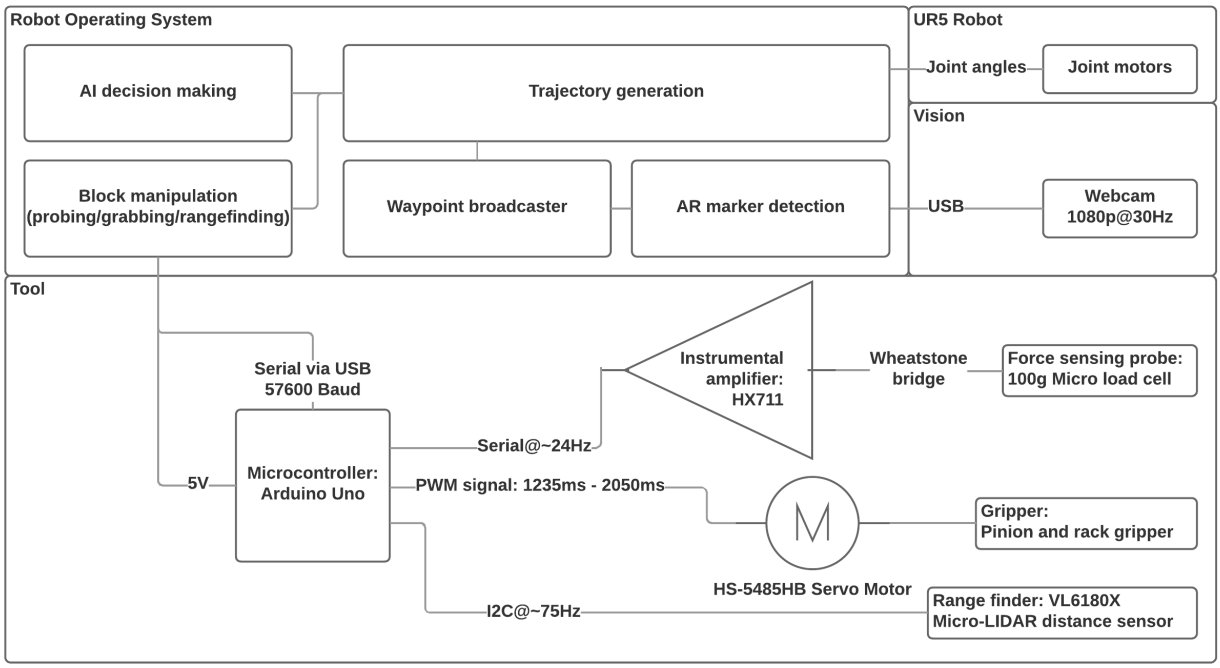
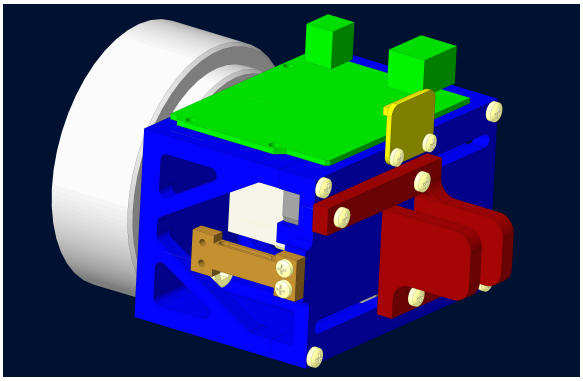
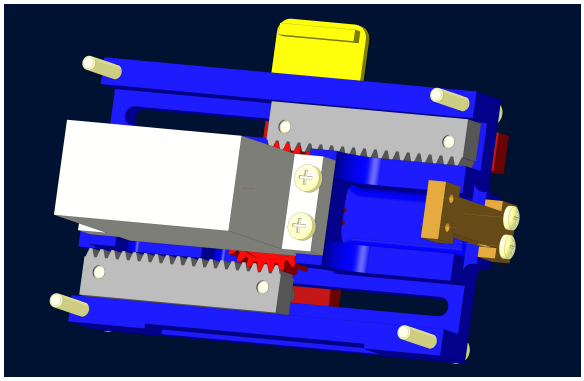


Fig. 1: The implemented system diagram



(a) Front view of the end-effector. The gripper is shown in red. The range finder is shown in yellow. The load cell is shown in brown, and the Arduino UNO is shown in green. The white round shape mount is the mounting surface on the robot



(b) Back view of the gripper with the mounting bracket off. The servo motor is shown in white, with a 32 pitch, 32 teeth gear attached, shown in red. The racks that are rigidly attached to the grippers are shown in grey.

Fig. 2: Solid model for the gripper mechanism assembly.

- **Sense looseness of blocks:** 100g micro load cell and HX711 load cell amplifier
  - According to this research article [3] on Jenga mechanics, a maximum force exerted to a block should not exceed 0.3N to prevent applying too much torque on top of the tower. Therefore, the sensor should have a threshold force of 30g.
  - Preferably, the measurements have to be quick and precise to about 0.1g to stop the robot in time if the force required to move a block is over the threshold. Therefore, in implementation, I used a 100g micro load cell for its precision.
  - Since the signal output for a load cell is a small differential voltage signal through a Wheatstone bridge, a specialized ADC is required to process and amplify the signal. I chose HX711 for this job because it has a readily available breakout board and it can run at up to 80Hz sampling rate.
- **Localize the salient block:** VL6180X Micro-LIDAR distance sensor
  - Since the gripper is a pinion and rack design, see the next item for details, it is crucial for the gripper to be at the center of the salient block so the robot will not harm the tower stability when pulling the block out. Therefore, the robot needs a way to know the middle point of the salient block. I chose to use a

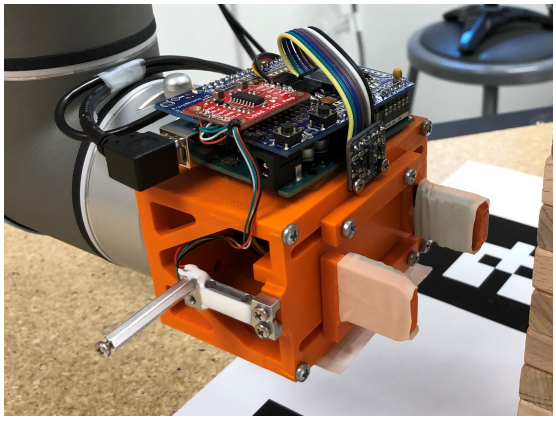


Fig. 3: The assembled end-effector on the robot

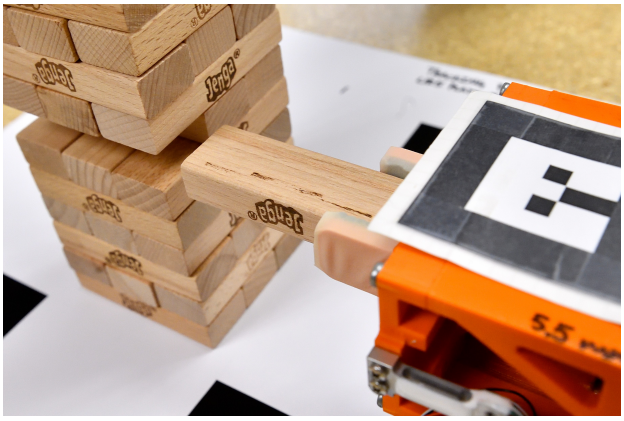


Fig. 4: The modification to the gripping surface provides excellent traction on the block. The AR marker in the photo for hand-eye calibration. Image Credit: JHU Homewood Photo - W. Kirk and J. VanRensselaer.

range finder for this purpose.
  - As the block will be pushed out 25mm from its original position, the range finder should have a 1mm resolution and a minimum sensing range of 10mm. It should have a low field-of-view so the change in readings will be fast and immediate when encountering a salient block.
  - A laser range finder is the only choice that satisfies the criteria above. More specifically, I picked VL6180X for its 5mm to 200mm range and about $15°$ FOV.
  - Implementation details on how to localize the salient block using the range finder are explained in the *Software Design* section.
- **Extract and return the block:** Pinion and rack gripper
  - For the robot to pull a block out and put it back on top of the tower, the gripper needs to have the ability to grip on the short side and the long side of a Jenga block, which are 25mm and 75mm, respectively.
  - Although there are readily available grippers to use in Wyman 170, they do not open wide enough to grip on the long side of a block.
  - Therefore, I designed a pinion and rack gripper with reference to a design that I found online [4]. See Figure 2 for a front and back view of the gripper mechanism.
  - I chose to use a servo motor since it can stop at any point in its range to account for small dimensional variations of each block, and with enough current it can hold a consistent torque so it would not loosen up when holding the block.
  - To further prevent the block from slipping off when the robot is moving, segments of heat shrink tubing are glued onto the gripping surface on the gripper to provide some deformation, and they are covered with surgical glove segments to increase friction. See Figure 4 for a photo of the gripper when it is holding a block.
  - The HS-5485HB digital servo motor for the gripper has a dedicated power source from a 5V-2A USB power outlet, and a 560pF decoupling capacitor is soldered close to the headers to smooth out the motor noise.
- **End-effector control:** Arduino UNO
  - Although a wide variety of microprocessors can be used, I chose to use Arduino UNO for its compact size and readily available libraries.
  - The board is running a modified ROS node firmware and communicates with ROS core on the workstation with UART at 57600 Baud. The node is modified to only have 128 bytes of input/output buffer to save dynamic memory space and increase stability of the program.
  - The microprocessor is directly powered by the 5V source in the USB.
  - All the electronics are connected to soldered headers on an Adafruit ProtoShield that sits on top of the Arduino board. Right-angled pin headers are specifically used to reduce the protrusion on top of the end-effector to prevent self-collision with the robot.
- **Grip change solution:** Double-tilted slope
  - A double-tilted slope for placing a Jenga block is designed to provide a consistent position for the robot to change the gripping position from the short side of the block to the long side, for when placing the block back on the tower.
  - The slopes are tilted $15°$ and $30°$; thus if a block is dropped on the slope, it will always end up at the same terminal position on the lowest point of the slope.
  - See Figure 5 for a photo of the slope with a block on it.

IV. Software

The software implementation is built upon a variety of existing packages and project specific packages that composed of more than 3000 lines of C++ code, with the majority in the trajectory control node, and some MATLAB and Python

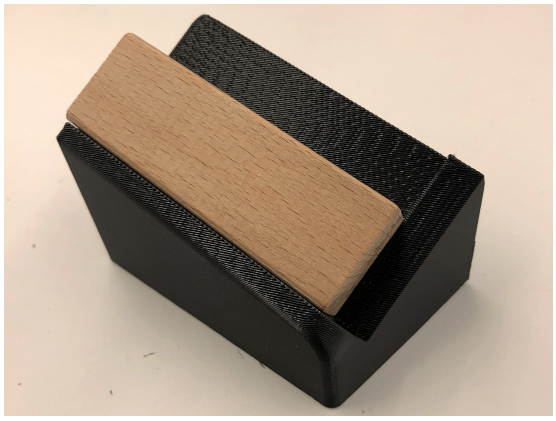


Fig. 5: The double-tilted slope with a Jenga block in its terminal position.

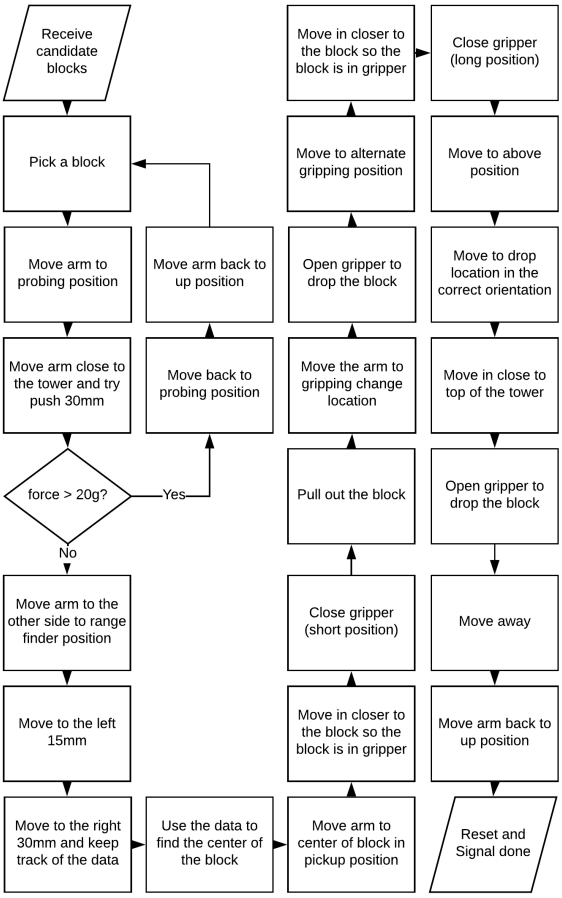


Fig. 6: The steps for generating a trajectory to play a block.

code. The MATLAB and Python dependency can be easily broken if the functionalities are rewritten in C++; they are used solely for ease of implementation for less complex parts of the system. See Figure 6 for a full flow chart for the trajectory control node to play a block.

A. Existing Packages

- Tower localization
  - *v4l-utils:* Linux driver for V4L usb cameras. Used to open and set parameters of the Logitech C920 usb camera.
  - *usb_cam:* ROS driver for V4L usb cameras. Used to drive the Logitech C920 usb camera.
  - *camera_calibration:* Used to find intrinsic parameters for the C920 camera.
  - *ar_track_alvar:* Track AR tags. Used for hand-eye calibration and Jenga tower localization.
- Control system
  - *industrial_core:* Provides an actionlib server named *follow_joint_trajectory* to control the robot along joint trajectory points.
  - *universal_robot:* Meta package for UR robots. Provides kinematics, urdf profile, and gazebo profile.
  - *ur_modern_driver:* A drop-in alternative for the dated ur_driver in the universal_robot package.
  - *rosserial:* Used to communicate with the Arduino-based enf-effector on the robot via USB.

B. Project Specific Packages

- *jenga_calibrate:* Provides camera calibration, hand-eye calibration routines, and AR marker detection launch files. Also includes AR marker related tf broadcasters. Detailed information is explained in the *AR Marker Detection* section below.
- *jenga_end_effector:* Provides *a-la-carte* functionalities for the Arduino based end-effector. Detailed information is explained in the *Firmware on the End-effector* section below.
- *jenga_launch:* Provides launch files to launch everything in the Jenga package family except jenga player, which intentionally needs to be launched manually by the human.
- *jenga_msgs:* Custom messages for the family packages.
- *jenga_player:* AI brain for the robot arm. Detailed information is explained in the *AI Player* section below.
- *jenga_ur5_control:* Provides trajectory generation and action execution for the AI player to interact with the robot. Also includes a tf broadcaster to publish waypoints based on the inferred tower pose. Detailed information is explained in the *Trajectory Generation* section below.

C. Software Design

1) AR Marker Detection: The implementation used a 1920×1080 resolution USB camera to detect multiple AR markers to infer the pose of the Jenga tower on the table using the *ar_track_alvar* package. As mentioned above in the *Hardware and Electronics Design* section, the AR markers are 90mm wide squares, and each marker is separated by 185mm to leave a 10mm white space around the markers and to put the 75mm wide tower in the middle of the marker array.

Since one or two markers may be obstructed from the view of the camera by the tower, the subsystem is designed to infer the tower location from one or more markers using prior knowledge of the tracking paper. i.e., each of the markers has

its own inferred tower location. In the case of multiple markers detected, these inferred locations are averaged to account for noise of individual markers.

The final tower location is averaged over previous locations using exponential moving average to smooth the change in tower pose over time, with $\alpha = 0.9$ in the first 100 time steps when the system detects a change in tower pose, and $\alpha = 0.1$ after that.

A similar exponential moving average is also implemented in inferring the pose of the double-tilted slope, which has its own AR marker.

2) Trajectory Generation: After the tower pose is inferred from the AR markers, waypoints are published around the tower and configurations to these waypoints are calculated by inverse kinematics(IK). Since there can be up to eight configurations for the same Cartesian coordinate, a heuristic based on relations of different joint angle pairs are employed to eliminate impossible and undesirable configurations. See Figure 7 for a screenshot of the waypoints in a simulation.

These waypoints are set up in a way that when the robot is traversing within these waypoints around the tower, it will not collide with the tower. The robot will stay within the waypoints until the final mile to its destination. The final mile is planned in Cartesian space using IK, exploiting the uniformity of the Jenga block. That is, as the waypoints on the side are defined at a certain block level, the robot just have to move down $15\text{mm} \times (level_{side} - level_{target})$ to reach the block level, and move to the left or right 25mm to reach the left or right block.

Once the robot is in position, it will execute predefined actions: probing, scanning, gripping, grip changing, and block placing. The trajectory generation node keeps track of the current game state to calculate the correct travel and block orientation from the waypoint above the tower when it is executing block placing action.

In the initiation phase, the trajectory generation node will execute a sequence of actions to self-calibration to gather information to compensate the noisy readings of the tower location. It will scan the two sides that the player node will play on using the laser range finder and calculate the center point of the tower and the average distance to the tower. The center point information is used to offset the expected block location to reach the actual block location accurately, while the average distances are used for the probe to reach the side of the tower accurately when executing probing action. After self-calibration, the robot will move to inferred drop and pickup position around the double-tilted slope. In this step, the human will have to confirm that the drop and pickup position will not collide with the slope. This is the only step that requires human intervention. Note that if the system detects a drastic change in tower location or orientation, it will execute the self-calibration sequence again to get accustomed to the new pose of the tower.

After initialization, the trajectory node will spin until it receives a target block to play from the player node. Then it will follow the flow chart in Figure 6 to try to play that block. Depending on whether the block is successfully played or is not removable, the node will report the resulting status back to the player and return to spinning for further instructions.

3) AI Player: There is inherently two sides that are easy for the robot to reach and two sides that are harder to reach due to how the robot is mounted on the table, as shown in Figure 9. Although the trajectory generation node can generate paths to all the sides, the ability to reach the harder sides heavily depends on the placement of the tower. Since the configuration in Figure 8a is close to the reach limit of the robot, and since the configuration in Figure 8b is close to the workspace boundary around the base of the robot, there may not be a full IK solution set for all the waypoints along the path to a target block on these harder sides.

To solve this problem, I implemented a Jenga player that distinguishes the easier sides and only send target blocks on these sides. By observation, the harder sides are always the side that is closest to the base of the robot, and the opposite side of it. Therefore, the player node would compare the distance of the side waypoints and determine the easier sides before publishing a target block to the trajectory generation node. On the rare occasion when there are two sides with equal distance to the base, $arctan2$ is applied to the two x-y coordinates, and the one that is smaller is considered the harder side. This calculation is repeated before each target block is sent; therefore, the player can adapt to the changes in tower pose, and switch to the appropriate game state for the new playing sides.

Due to clearance issues between the robot and the table, the AI player only issues target blocks from level 6 and above. That leaves 12 levels for the robot to play on, and that is 6 levels on one side. To reach the objective of playing 6 blocks, the player can not just play the middle blocks, which are the most stable blocks to remove, as it is possible that the middle block is not removable and in such case, the robot may not be able to extract 6 blocks from the tower without the human rotating the tower. By observation, if the middle block is not removable, then it is very likely that the blocks on the sides are very easy to push. Therefore, to maximize the blocks played, the agent will first attempt to play the middle block. If the robot fails to play the middle block, i.e., requires forces over the threshold to move, then the player will try the blocks on the sides.

4) Firmware on the End-effector: The microprocessor board selected for the project, the Arduino UNO, have only 2k dynamic memory. By the structure of an Arduino program, if a variable is to be shared between setup and loop functions, then it must be global. ROS node that run on the microprocessor board occupies a large amount of space in the dynamic memory, and the firmware will not fit without tuning the buffer size and the number of publisher/subscribers for the node handle object. After numerous tests, a 128 byte input and output buffer size and 4 publisher and subscriber count are the minimum stable parameters for the node handle. Further consolidated message types for various control and sensor messages, the global variables occupy about 75% of the dynamic memory and the firmware runs stable across multiple sessions.

To maximize the publish rate of force messages from

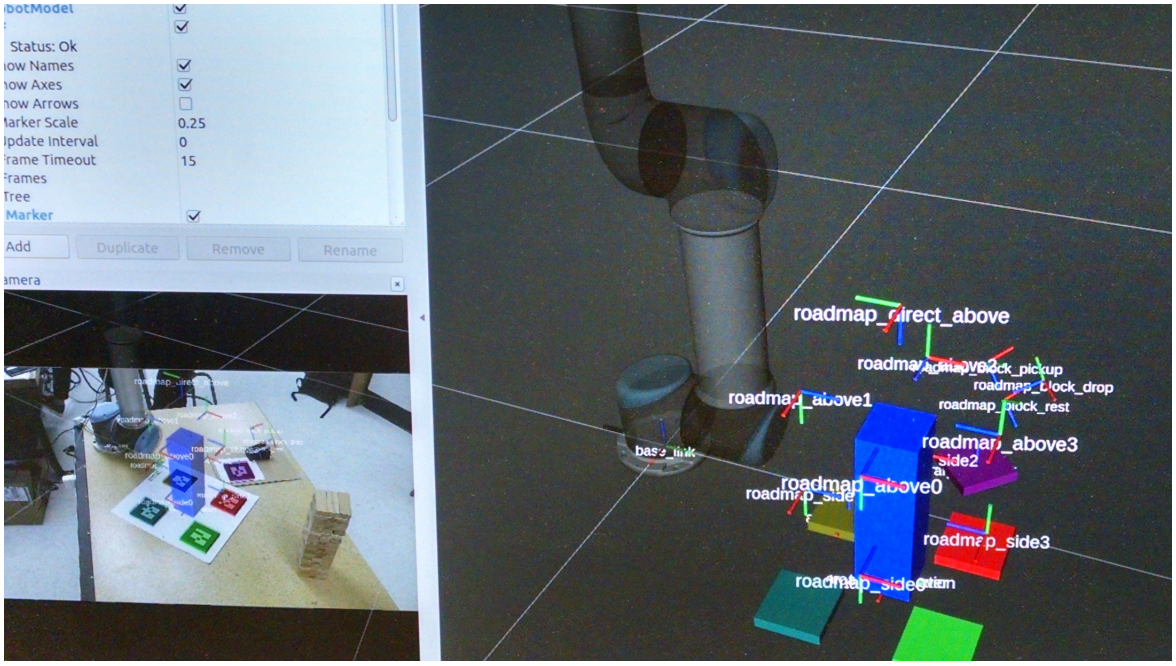
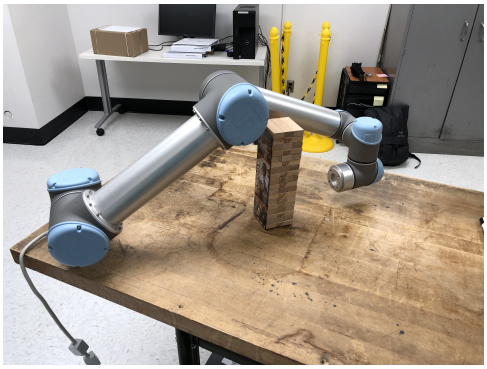
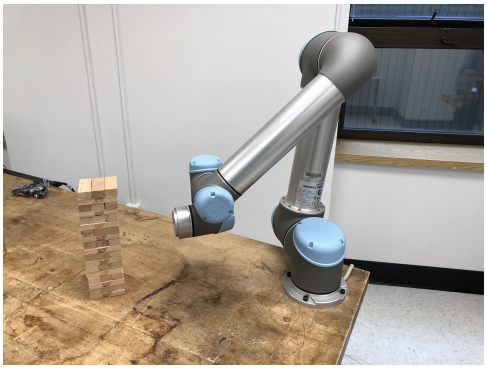


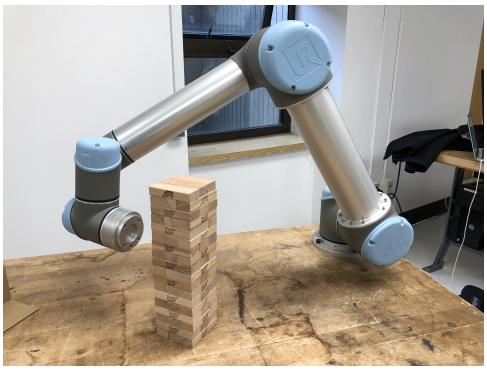
Fig. 7: A screenshot of AR tags, inferred tower pose, and the waypoints around the tower.



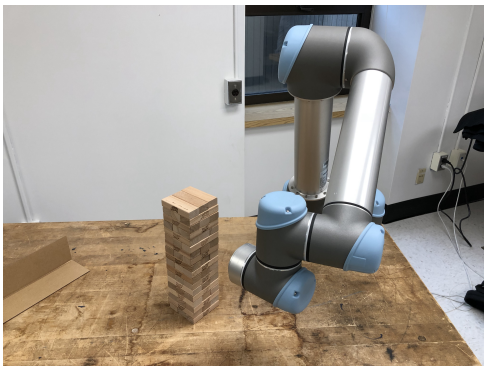
(a) A configuration to reach the far side of the tower.



(b) A configuration to reach the near side of the tower.



(c) A configuration to reach the right side of the tower.



(d) A configuration to reach the left side of the tower.

Fig. 8: Photos of actual robot in different configuration around a full size Jenga tower.

the load cell and range messages from the range finder, the electronics are turned on and off on demand, ensuring only one component is turned on at a time, and that each component gets maximum processor time possible. The result is the range finder publishes range message at around 75Hz, yielding around 800 data points in a single block scanning action; the probe publishes force message at about 24Hz, and the robot can stop itself in 1mm if the force on the probe is over the threshold.

5) Salient Block Localization Using Range Finder: Ideally, when the range finder encounters the boundaries of the salient block, the readings should be reminiscent of a step down function followed by a step up function. By direct observation, there are two cases when the range finder is scanning. The first case is when scanning the side blocks. The range finder data will have large change in readings when reaching the side of the tower, and a moderate change when reaching the middle part of the tower. The second case is when scanning the middle block. In this case, the readings will have two moderate changes.

Although a low FOV laser range finder is used, the data collected in range finding action are still more of a ramp function when the readings change. See Figure 9a for the raw readings and Figure 9b for the smoothed data. However, after differentiating the data points, two clear peaks present in all sessions. These are the points where the range readings are most rapidly changing, which roughly translate to the boundaries of the salient block. See Figure 9c for the smoothed differentiated data.

In dawn of this observation, I have implemented the localization function to first smooth the data collected in the range finding action with a moving average filter with size 29 window. Size 29 is selected from experiment to yield better results without over smoothing. Then the data will be differentiated, i.e., two consecutive data points are subtracted. The returned data will have a max peak and a min peak, and the program will consider these two points as the boundary of the salient block. The transforms of the range finder at these two points are queried, and the compensation is calculated as the difference from the average of y-axis translation of these two points to the expected middle point as inferred from the waypoints.

V. Results

As shown in the introductory video [5], the robot can play a Jenga block autonomously in roughly 60 seconds. However, due to stochastic nature of how the blocks compose the tower in each game, it is not guaranteed to extract at least six blocks in every game without needing to rotate the tower to the other orientation by a human. In this uncut video [6], the robot successfully played more than 6 blocks without human intervention.

In an unrecorded session, the robot reached the highest level of 21, three levels in addition to the starting 18. That is, nine blocks was extracted from the tower.

VI. Observation and Future Improvements

In the playing sessions, I found that the force required to move the blocks is different from level to level: the higher blocks require much less force to move, and when the block is moved, it is more likely to move the blocks above it all together. This is probably due to the lower blocks have more weight above them and thus is harder to move, while the higher blocks have merely one or two levels of blocks above them, so the static friction alone will be strong enough to move the blocks above them when trying to push a block out. One possible solution is to change the force threshold from a predefined constant to a function of the level a block is at. The higher a block is, the lower the force threshold for the robot to stop probing the block. This is possible because the 100g load cell used on the tool is precise enough to have 0.1g resolution after calibration.

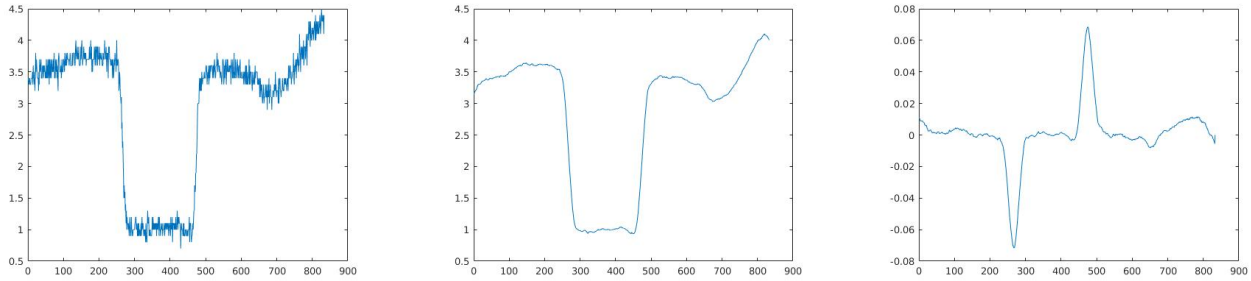
Another issue found when the robot is playing Jenga is that if at a certain level the side blocks are removed, then it becomes easier for the tower to rotate when a side block at a higher level is probed and pushed. This is normally not a huge problem because the robot will likely push the tower back as a byproduct of executing scanning and gripping actions. However, at times this may cause the tower to distort and collapse prematurely. I identify the cause of this phenomenon to be the remaining middle block provides a pivot point for the tower above it to rotate, while a full level is more stable rotation-wise because the friction force between the blocks on the same level counteracts the tendency to turn. This problem could be avoided if the robot plays the higher blocks first, instead of the current strategy of playing lower blocks first. This ensures the lower parts of the tower is full and more resistant to rotation.

The project can also be extended to have computer vision incorporated to the system that analyze the tower composition, so the robot can play against a human. Since the blocks are rather plain, some small round stickers can be stick onto each block to create features for the computer vision program to analyze. A computer vision system can also be used to detect whether a tower is twisted so the robot can execute appropriate actions in attempt to push the tower back to proper position.

The AI player node is written in a modular way for future addition of different agents. One possible implementation is an agent that decides what block to play based on the tower composition and a Hidden Markov Model for the Jenga tower. This way the robot will not have to poke each block in a level to determine what is playable, but probe only the most possible blocks.

VII. Lessons Learned

- When 3D printing, leave enough material between the walls of holes and the edges of parts so the holes will not crack when drilling the holes in preparation for tapping.
- If possible, use a heat gun to heat up the screws and the part before tapping to soften up the plastic material.
- Extra care is needed for soldering 30AWG wires onto a PCB. Use extra wires as strain relief or hot glue the connections in place.



(a) Raw data (b) Smoothed data (c) Differentiated data

Fig. 9: Graphs of data points from a range finding action on a middle block performed in a game. The data is smoothed with a moving average filter with size 29 window.

- It is always a good idea to separate motor power and logic power. Servo motor can draw almost 2A current and a sudden variation in current may damage the microprocessor.
- Add decoupling capacitors in parallel with the power of motor and ground to separate the noise from the motor.
- Use add_dependency in CMakeList.txt to ensure messages and services are generated before a node is compiled. Command: *add_dependencies(**[node name]** ${**[package name]**_EXPORTED_TARGETS})*
- ROS programs by default are single-threaded, and callback functions will not be processed until ros::spin() or ros::spinOnce() is called.
- If you designed the AR tag to be at a known transform relative to base_link, then the calibration process will be much easier, because now the transformation from base_link to marker and from camera to marker are known, and transformation from base_link to camera is just a calculation away.
- AR markers need white margins on all sides for the camera to detect them properly and consistently.
- Quaternions can not be averaged. To average quaternions, you will need to do matrix multiplication and eigenvalue decomposition. On the other hand, RPY can be averaged.

ACKNOWLEDGMENT

The author would like to thank Prof. Louis Whitcomb, Prof. Pedro Julian, and Prof. Simon Leonard for their kind support and expansive knowledge on subjects related to this project. The author also gratefully acknowledges course teaching assistants Bayo Eisape and Taylor Paine for their valuable help and suggestion on the project.

ADDITIONAL MATERIAL

The project is available on GitHub as a public repository: *https://github.com/RexxarCHL/UR5_Plays_Jenga*

REFERENCES

[1] T. Kroger, B. Finkemeyer, S. Winkelbach, L. O. Eble, S. Molkenstruck, and F. M. Wahl, "A manipulator plays jenga," *IEEE Robotics Automation Magazine*, vol. 15, no. 3, pp. 79–84, 2008.

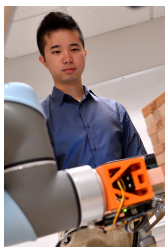
[2] T. Kroeger, "A robot plays jenga." [Online]. Available: https://youtu.be/yD63f7MKPjI

[3] J. Ziglar, "Analysis of mechanics in jenga," 01 2006.

[4] papabravo, "Rack & pinion robotic gripper jaw." [Online]. Available: https://www.thingiverse.com/thing:2661755

[5] C.-H. Lin, "Ur5 plays jenga." [Online]. Available: https://youtu.be/KtRhFWFU7mw

[6] ——, "Ur5 plays jenga - uncut version." [Online]. Available: https://youtu.be/0ORZyjbzYnw



Chia-Hung Lin
Chia-Hung Lin is a Robotics MSE student in the Johns Hopkins University. Image Credit: JHU Homewood Photo - W. Kirk and J. VanRensselaer.