VX Lab's RoboCoStack Final Report

Joseph Sierd David Williams

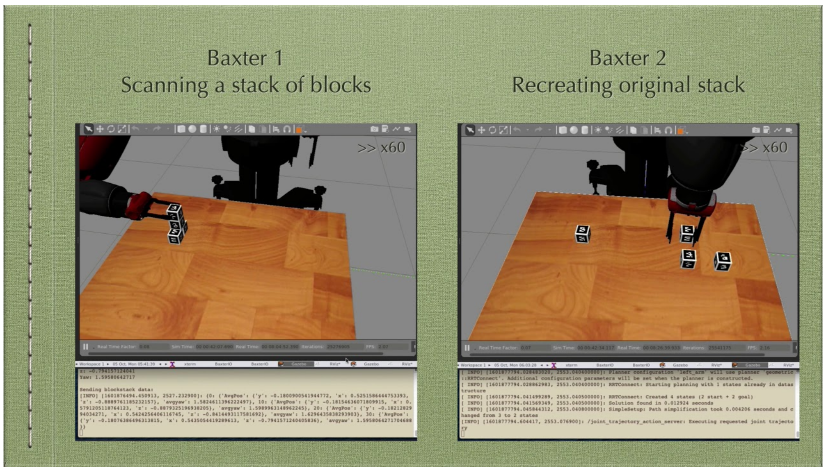
Supervisor: Ian Peake,
Laboratory Manager of VXLab, RMIT

8 November 2020
RMIT University, Melbourne

Table of Contents

Project Summary

Click on the below image for a demonstration video of our project:



```
z: -0.794157124041
Yaw: 1.59580642717

Sending blockstack data:
[INFO] [1601876494.450913, 2527.232900]: {0: {'AvgPos': {'y': -0.1800900541944772, 'x': 0.5251586444753393,
'z': -0.8889761185232157}, 'avgyaw': 1.5824611396222497}, 10: {'AvgPos': {'y': -0.18154636071809915, 'x': 0.
5791205118764123, 'z': -0.8879325196938205}, 'avgyaw': 1.5989963148962245}, 20: {'AvgPos': {'y': -0.18212829
94034271, 'x': 0.5424256406116745, 'z': -0.8414493117581692}, 'avgyaw': 1.6296435838293903}, 30: {'AvgPos':
{'y': -0.18076386496313815, 'x': 0.5435054419289613, 'z': -0.7941571240405836}, 'avgyaw': 1.5958064271704688
}}
```

```
[ INFO] [1601877794.028483020, 2553.040400000]: Planner configuration 'left_arm' will use planner 'geometric
::RRTConnect'. Additional configuration parameters will be set when the planner is constructed.
[ INFO] [1601877794.028862983, 2553.040400000]: RRTConnect: Starting planning with 1 states already in datas
tructure
[ INFO] [1601877794.041499289, 2553.040500000]: RRTConnect: Created 4 states (2 start + 2 goal)
[ INFO] [1601877794.041569349, 2553.040500000]: Solution found in 0.012924 seconds
[ INFO] [1601877794.045844312, 2553.040800000]: SimpleSetup: Path simplification took 0.004206 seconds and c
hanged from 3 to 2 states
[INFO] [1601877794.604417, 2553.076900]: /joint_trajectory_action_server: Executing requested joint trajecto
ry
```

RoboCoStack is a collaborative robotics project that builds on previous infrastructure developed for Baxter, a multi axis two armed robot.

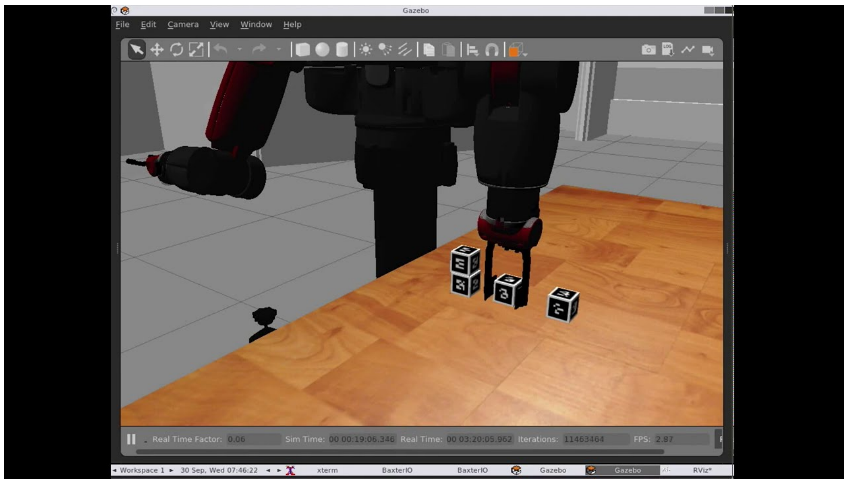
A vision based system was developed for scanning block stack arrangements and sending this information to a second robot to them mimic the original stack. The implementation assumes the use of a scanning robot and a stacking robot as two way communication has not been enabled.

By placing the scanning arm of the first robot into an optimal position to view the block arrangement, location and orientation data is collected and then shared. Once the second robot receives this data, it is sorted by z-axis and then stacked one by one into the original arrangement.

At present there is no collision avoidance implemented when moving blocks, however the stacking algorithm has been implemented in such a way to minimise the risk of collision between objects to near zero.

System Description

The system uses Gazebo[1] to simulate the actual VX Lab at RMIT. Baxter, a multi axis two armed robot, has been placed in front of a table with 4 blocks wrapped in ALVAR[2] markers for identification. Manipulation of the arms is done using the MoveIT[3] framework with a combination of path planning and waypoints depending on the action. Communication between the two robots is achieved using ROS Topics[4].


Video demo of block scanning and stacking

Click on the image above to see a video demonstration of Baxter scanning and stacking a tower of 4 blocks.

All code is contained within the directory: baxter-simdemo/rosie/moveit. To set up the demo, the file set_arms_to_scan.py takes a series of arguments that match various block stack arrangements. Each argument places the scanning arm into the optimum position to map the locations of each block arrangement. For example:

./set_arms_to_scan.py yaw

will match the spawn script inside the spawnstack file with the section commented: '# 4 in a row with yaw'. For a full description on how to set up and run the demonstrations see the section titled: 'How to build and run on a VX Lab blade instance' below.

The algorithms for scanning and stacking are contained in blockstack_alvar_demo.py and the methods for moving the arms for scanning

1. http://gazebosim.org 2. http://wiki.ros.org/ar_track_alvar 3. https://moveit.ros.org 4. http://wiki.ros.org/Topics and stacking are all contained within moveit_baxter.py. Within the init() method in moveit_baxter.py the max acceleration for the limbs is set at 0.5. If you wish to increase the speed at which the arms will move then this is where you change that. Currently 0.5 has been selected to balance speed and reliability as a higher number causes greater joint failures to to sudden movements. To change the speed of the simulation itself you can modify the values in the physics tag in baxter-simdemo/rosie/vxlab.world file. However this will alter how the models interact, a faster simulation causes block drift which causes issues with maintaining stack arrangements.

Deployment Guide

How to build and run on a VX Lab blade instance

```
git clone https://github.com/s3569541/baxter-simdemo.git
cd baxter-simdemo
git checkout branch blue-mir100
```

On first time run only:

```
docker-compose build
./create-vxlab-network
```

To run two sims side by side:

```
docker-compose -f docker-compose-two.yml up
```

This will give you series of containers, the two we are interested in are:

gazebo and gazebo2

To view simulation:

ip_address_of_blade:8080/vnc_auto.html

Scanning Baxter (gazebo container):

```
docker exec -it gazebo bash
DISPLAY=novnc:0 gzclient                    (view the simulated world)

docker exec -it gazebo bash
DISPLAY=novnc:0 rviz                        (see Baxters view of the world)

docker exec -it gazebo bash
cd moveit && source ./init                  (startup the moveIt framework)

./set_arms_to_scan.py       init (on start up only: moves left arm out of the way and
opens grippers)
./set_arms_to_scan.py yaw                   (set scanning arm to ideal position)
./spawnstack                (uncomment only yaw instructions before execution)
./blockstack_alvar_demo.py primary
```

Stacking Baxter (gazebo2 container):

```
          docker exec -it gazebo2 bash
          DISPLAY=novnc:0 gzclient              (view the simulated world)

          docker exec -it gazebo2 bash
          DISPLAY=novnc:0 rviz                  (see Baxters view of the world)

          docker exec -it gazebo2 bash
          cd moveit && source ./init            (startup the moveIt framework)
          ../twin-relay                 (communication from gazebo to gazebo2)

          docker exec -it gazebo2 bash
          cd moveit && source ./init            (startup the moveIt framework)
          ./set_arms_to_scan.py         (on start up only: moves right arm out of the way)
          ./spawnpickup                         (sets pickup blocks to ideal position)
          ./blockstack_alvar_demo.py stack (stacks blocks based on received data)
```

When finished: docker-compose down --remove-orphans

Developer Guide

All scripts are contained in: baxter-simdemo/rosie/moveit/

Models for the table and cubes are located in: baxter-simdemo/rosie/ in the directories markerblock_1 to 4 and markertable. The configuration of the models is in the model.sdf file and the wrapper images are in the materlies/ textures directory. To change the image wrapper you can simply swap out the .png file but the new file must have the same name as the original as the mesh file (that defines the shape) located in the meshes directory relies on this name.

The directory of each model must remain within the rosie directory for them to spawn correctly and I have found that the name tag referenced within the script tag of each model.sdf should be unique between models otherwise the wrappers wont display as intended or you end up with multiple cubes that are the same rather than unique.

locallib.py contains everything related to the ALVAR system and translating poses between camera frames for use in location. This was developed in collaboration with the technical manager for use by the moveIT system.

moveit_baxter.py contains the methods developed for moving Baxters limbs. init() needs to be called before using any of these. init_left_gripper() only needs to be called if you intend to use the gripper. This can easily be modified to initialise the right gripper.

In order for locateBlock() or pick() to work, the required block must first be in view on the camera. pick() will automatically move progressively closer to required block to gain the needed accuracy to pick up.

move_arm(), drop_arm(), raise_arm() are used internally only for higher level methods.

blockstack_alvar_demo.py is the scanning and stacking script as determined by sys.argv, use 'primary' to scan and send data, any other arg to wait for data and then stack. No arg is for a single robot implementation.

./spawnpickup will create a table and 4 blocks in optimum position for the pick() method above to locate a block for the left arm.

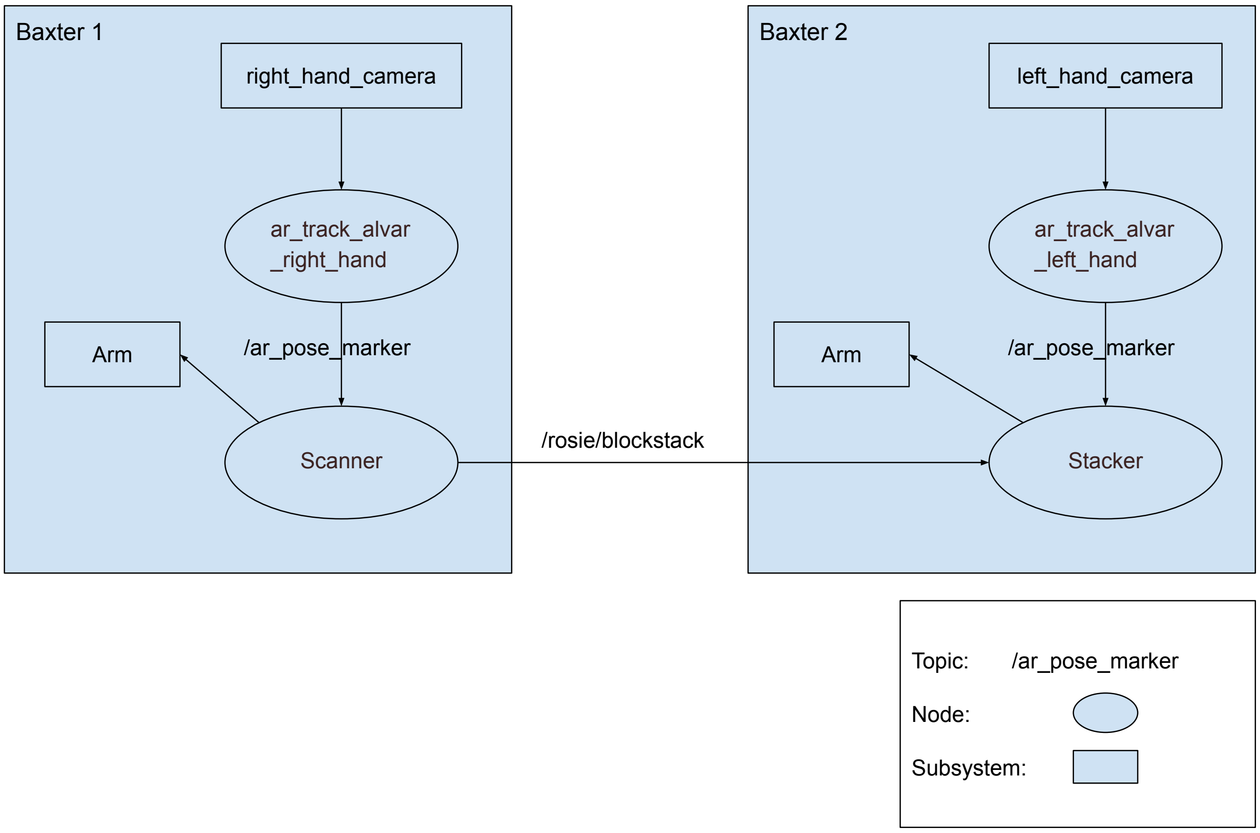
./spawnstack needs to be paired with **set_arms_to_scan.py**. The arguments for the second script need to match the spawn instruction uncommented in the first as each arm positing in set_arms_to_scan has been calibrated for each pre defined spawn script.

Use **set_arms_to_scan.py** with no args if you simple wish to move the arms out of the way as a result of collision or error.

Depending on the path taken, you may need to use **./spawndelete** before calling **set_arms_to_scan.py** to a different position as the arm may collide with the table and fail.

Technical Solution

Runtime architecture



System Design

All code from the project is contained in baxter-simdemo/rosie/moveit/ on the blue-mir100 branch.

Baxter 1 - Scanning

Baxter 1 is the scanning robot. Image data from the right_hand_camera is passed into alvar marker recognition system and published on the ar pose marker topic. This is used by the scanner subsystem that iterates over block numbers storing position and orientation data for each block within the field of view of the right hand camera. The arm position of Baxter 1 is set in a way to maximise the accuracy of the location data of each block.

Once collected the data is then published on the /rosie/blockstack ROS topic.

Baxter 2 - Stacking

After initialisation, Baxter 2 listens on the /rosie/blockstack topic, once data is received this is then converted and sorted by z axis values of each block to ensure correct stacking order of the blocks.

Baxter 2 then uses the left_hand_camera image data translated by the alvar marker node to locate required blocks to be moved into position.

The pick method first places the left arm of the stacking robot in a position known to be able to view all blocks produced by the spawnpickup script. It then progressively moves closer to the required pick up block taking fresh position readings along the way. This is to ensure accuracy of the data which reduces collisions and failed pick ups. When the arm is directly above the required block, block pose data is taken again and a series of waypoints is calculated between the position of the arm and the final grab pose. This has ensured almost no collision occur when moving to the grab position. After the grippers close, the arm then lifts. This is to clear the arm of any nearby blocks to reduce collisions between remaining blocks when the place() method is called.

When place() is called, the arm moves to just above where the block needs to be and drops down tp the release position and then lifts again. These calls to drop() and raise() have been implemented to reduce collisions between the arm and neighbouring blocks.

Test Management

We used manual testing in this project through the use of the Gazebo simulator and Rvis to understand the robots perspective of the world.

Automated testing was impractical for this project due to the variability in the interactions between the components of the system. In developing the project it was found to most useful way to gain insight into problems with the script was by watching the simulation run, interactive develop the function and work towards reproducibility. This way we could identify problems early and work through them. This also gave insight into the impact of changing one aspect of the project had on other systems.

A test case is available in ~/rosie/ikeg/block_pickup_test.py that will verify that a block has in fact been picked up which was used in the early stages of the project.

Validation of end goals was done through the production of videos of Baxter completing assigned tasks such as collecting location information with one arm of a stack of blocks and using this information to manipulate a set blocks with another arm into the same configuration. There by validating that not only the scanning algorithm was accurate but also the stacking algorithm.

The threshold by which to determine if a goal was successfully met was repeatability. If a given stack that was scanned could reliably be reproduced to resemble the initial stack, with no collisions or failed pick ups and that the final structure would hold with out falling over after being created at least 3 times in a row, then this was deemed to be a success and we could say we had successfully implemented the goal of using two robots to scan, send and stack a set of blocks.

This was successfully achieved and the current implementation reliable achieves scanning and stacking multiple stacks on runs greater than 3 in a row.

This entire project will be reused for testing purposes by future VX Lab projects. Assets such as the table and Alvar blocks that were created during this project will be used in future simulation projects along with the moveIT implementation which will allow for comparison of robot arm manipulation against the original system as future group will to extend upon this and previous projects.

Development History

The Project was developed using fail fast iterative development. Find problems early to guide creative problem solving. We worked in close collaboration with the VX Lab technical manager to extend the framework to incorporate new features and ensure that what was developed could be reused for future projects.

As we were doing this project entirely in Gazebo we had to start with creating the models for Baxter to use. The initial code base already had a set up for baxter it self and a simulated layout based on the VX Lab, we needed to create models to interact with. This involved following many online tutorials around created models for use in Gazebo. A good starting point was to use a GitHub repo (https://github.com/osrf/gazebo_models) with pre defined models and spawn them in gazebo. This facilitated learning how model.sdf files worked in collaboration with mesh files and how to modify them to suit out needs.

In developing the codebase, repeatedly running scripts allowed us to understand how the scripts interacted with the robot and with the simulated world. Often having to use click and drag to manually move Baxters arms to predefined locations to get data about how the relative positions of each camera worked in relation to each other and the block we were trying to manipulate.

Understanding the variability between runs was also important to help guide how scripts were refined to improve repeatability and therefore reliability of Baxters movements. Rvis was very helpful with this as it have us information about how Baxter saw the world which helped to debug code and also develop methods that could deal with inaccurate arm movements, joint failure and variable camera readings.

Stage 1: Creating Models

In the end we settled on using Blender modelling software to create a cube and wrap it in an ALVAR png image file that gave us the required markers for location and orientation detection and distinction between block id's.

The physics of these models was an on going issue. As the project involved moving and stacking the blocks, the physics of the models themselves as well as the interaction between models was very important and was an on going issue throughout the project.

This was first encountered when initially spawning one block on top of another. It slide right off as if there was no friction. The first obvious thing to try was set the frication variables in the surface tag of the model.sdf file for each cube. This had no effect. After much research and trouble shooting a solution was found setting the contact and friction variables inside the surface tag to appropriate variables so the blocks could act like solid blocks with reasonable friction. This combined with inertial values for mass and inertia resulted in acceptable model behaviour for our purposes, however sim drift continued to be an issue where the blocks were never entirely at rest regardless of how the above variables were set.

This caused problems for stacks higher than 2 blocks as they would fall quickly and was not resolved until the end of the project where it was discovered that modifying the physics tag in .world file for gazebo which increased that rate of the time_step while decreased the size resolved the drift and we were now able to create stacks of blocks at least 4 high with no block drift causing problems. The disadvantage of this however was that it significantly slowed the simulation due to the smaller and more frequent time step and cause the script to take significantly longer. This was deemed acceptable however as we required the accuracy of the new simulation runs for repeated runs.

Stage 2: ALVAR System

As mentioned previously, the pre existing ALVAR marker recognition system was used for locating blocks. Initially this was able to locate a single marker on a block. To distinguish one block from another you could infer block id's by the marker number. Marker ID's 0 to 9 was block 0, 10 to 19 was block 1 and so on. So we have block number 0, 1, 2 and 3. The camera was able to detect not only number and position but also orientation. So we could infer if it was a top or side marker.

This system was later extended in collaboration with our supervisor such that if you call getAvgPos(block_id) from locallib.py the improved method could infer on its own, based on any visible marker the block id and centre of mass. The result was more accurate data that made development of scanning and stacking algorithms much more efficient.

Stage 3: ROS MoveIT

The start up code used inverse kinematics solver for arm movement. A goal for this project was to implement MoveIt motion planning. This combined with the ALVAR system would allow us to develop automated scripts to locate and stack blocks to a pre defined data structure.

The result is a file with defined methods to have Baxter complete tasks. Import moveit_baxter.py into a script file and you call locateBlock(camera, marker) to get location data from a specific camera for use in scanning. There are also pick and place methods that allow you to pick up a block if it is within the field of view of the camera, and the place it at any given location.

This frees up the script file to only deal with scanning and stacking using high level methods. Which allows for greater flexibility and adaptability to use the moveIT framework for other purposes

In the development of these high level methods, a number of approaches were taken to increase the reliability in locating and moving blocks. The accuracy of the location data was relative to the distance of the camera to the block. So the arm is set to an initial location where all blocks are visible and then it will move to just above the required block to re take location data before descending to grab the block. This approach means there are few to almost no collisions between the gripper and block when moving to the final grab position which significantly improves repeatability of the scripts.

Most movements use the moveit_commander planning system which works well for large movements. We ran into consistent issues however with small movements. The end solution for this was waypoints to compute the final path to grab which increased reliability of the final path to pick up the block.

Review

The project goal was to achieve two independent robots working collaboratively to maintain and update a set of blocks based on changes to one of the stacks from an external force.

This was simulated using two independent gazebo environments. While we didn't achieve full two way communication between the two robots, we did implement a scanning and stacking robot. Working closely with the client through out the project, all changes to the initial specification were discussed and agreed upon and the client deems the current final implementation a success to what we set out to achieve.

Working in an often unpredictable environment like simulated robots is a fluid, there are always limitations and problems you can’t predict. Through the course of this project we successfully overcame many of these challenges and negotiated deviations to the project priorities when challenges were not able to be overcome. All to the satisfaction of the client.