Project 3 Final Report

Robot System Control for Automating Mosquito Microdissection

EN 601.656 Computer Integrated Surgery II

Zhuohong "Zooey" He
(zhe17@jhu.edu)

February 2021 to May 2021

1 Acknowledgements

I would like to thank our industry partners at Sanaria Inc. for their continued support on our project. Their goal of "malaria eradication through vaccination" will save countless lives that would be lost to the malaria epidemic. I am grateful to be a part of this vision for a brighter future.

I would also like to thank my advisors Dr. Simon Leonard and Dr. Russell Taylor of the Laboratory for Computational Sensing and Robotics (LCSR) at the Johns Hopkins University. Their advice and guidance were critical to the success of my project. I appreciate their time and enthusiasm at our weekly update meetings which have made a positive impact on my progress.

Lastly, I would like to recognize my other teammates, namely Wanze Li and Balazs Vagvolgyi, for their support during the debugging, testing, and documentation phases of my project.

2 Executive Summary

The malaria epidemic infects over 228 million individuals worldwide through mosquito agents, and the development of a vaccine could significantly reduce the death and suffering in many developing countries. [7] Sanaria Inc. is working on a promising vaccine with proven effectiveness towards preventing malaria infection. [3]. A major bottleneck in vaccine production is harvesting mosquito salivary glands. Our team at JHU is working on a robotic system to automate gland harvesting. For my CIS II project, I worked on a control algorithm which incorporated error detection, error recovery, and parallel process to support a new iteration of our robot system. I created controllers for three independent nodes, conducted successful testing, and made important contributions to improving the system.

Contents

3 Introduction

Malaria vaccine production and distribution is a critical public health challenge for many developing countries. Sanaria Inc. is working with the Johns Hopkins University Laboratory for Computational Sensing & Robotics (JHU LCSR) to develop a robotic system to overcome a bottleneck in vaccine production by automating mosquito saliva gland collection. Previous researchers at JHU have successfully extracted salivary glands by developing the robot hardware, control algorithm, and computer vision models. My goal will be to implement improvements to the existing control algorithm to increase parallelisation, add error recovery, and support the next generation hardware and computer vision efforts.

4 Background and Significance

The malaria parasite is responsible for a global disease affecting 228 million individuals and causing 405,000 deaths in 2018. [7] Malaria is most prevalent in underdeveloped countries where controlling the spread of the parasite is a national public health challenge. The parasite is transmitted through mosquitoes as they ingest blood from an infected person and pass the parasite to another person . Prior to transmission, the sporozoite version of the parasite resides in the mosquito's salivary gland. To curb infection, Sanaria Inc is developing a vaccine which has proven to be highly effective with long-lasting protective effects against malaria. [3] However, extracting the salivary glands from recently deceased mosquitoes is one remaining bottleneck in vaccine production currently performed by human operators. If extraction is automated, Sanaria will be able to significantly reduce the infection rate across the globe and save the lives of countless individuals living in areas where malaria is prevalent.

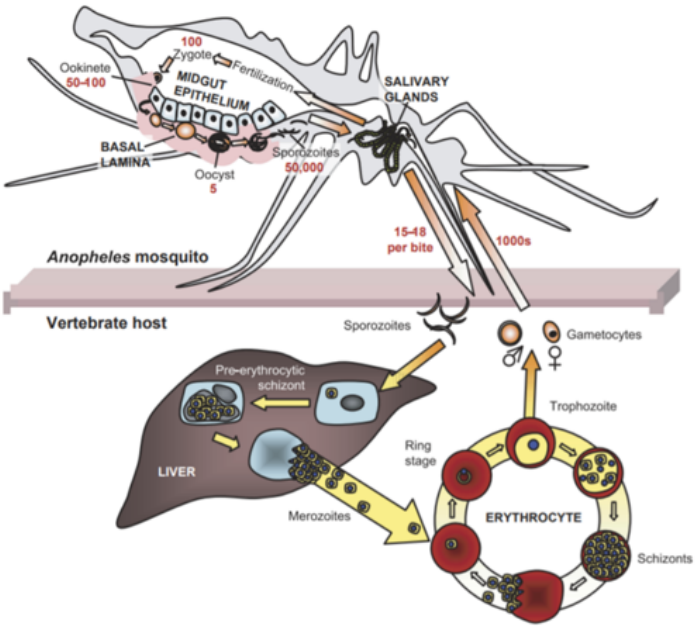

Figure 1: *Malaria life cycle. Note the position of the salivary glands within the mosquito upper abdomen and its function of storing the sporozoite immediately before infection into the vertebrate host.[5]*

Manual, operator-driven, saliva gland extraction has three steps which can be visualized in figure 2: (1) Remove the mosquito head using the edge of a hypodermic needle as a knife, (2) squeeze the thorax to extrude salivary gland, (3) separate and collect the salivary gland.

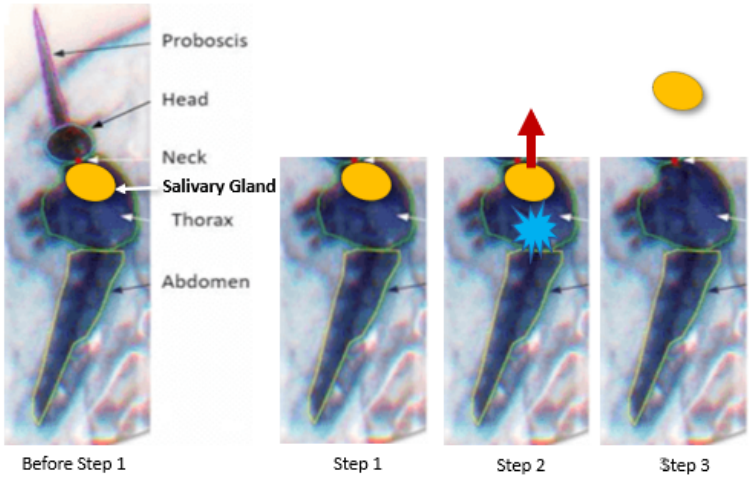

Figure 2: *Three steps associated with extracting a mosquito's salivary gland (yellow).*

5 Prior Work

Two systems have been developed to automate mosquito dissection. First, M. Schrum et al. developed the semi-autonomous mosquito microdissection system (sAMMS) (fig 3) in 2018 which still required a human operator albeit increasing the dissection rate from 300 mosquitoes per hour (mph), to 430mph. Our goal dissection rate is 600mph. [5] [6]

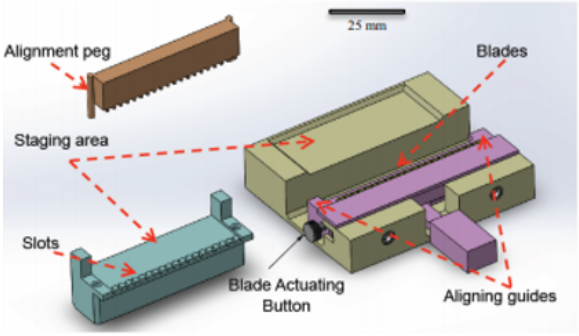

Figure 3: *The semi-autonomous mosquito microdissection system (sAMMS) which was tested for use by trained human operators at Sanaria Inc. [5]*

A second system started by H. Phalen et al. in early 2020 focuses on full automation through a 4 degree of freedom (DOF) robot manipulator with a controlled gripper as the end-effector. [4] In another paper accepted for the ICRA 21 conference, the team at JHU has incorporated further upgrades to Phalen's work. New hardware including a squeezer, turntable, linear stage, and cleaning station were added to increase the completeness, autonomy, and speed of the system. This system starts with the 30-40 mosquitoes in a center dish (fig 4). Next,

an operator drags mosquitoes onto the turntable's loading area (fig 5a. The turntable then rotates the mosquito until it is under the overhead camera, in front of the robot gripper. The robot and gripper uses the camera-fed convolution neural network (CNN) output to execute a placement between the blades of the cutting apparatus. The apparatus then decapitates the mosquito and the cartridge moves the mosquito to the squeezing station which applies pressure to the abdomen and extrudes the salivary gland from the neck. Our experiments demonstrate that one mosquito is processed every minute. This results in 60mph, a speed significantly lower than our 600mph goal, yet it demonstrates the feasibility of the concept and the areas of improvement.

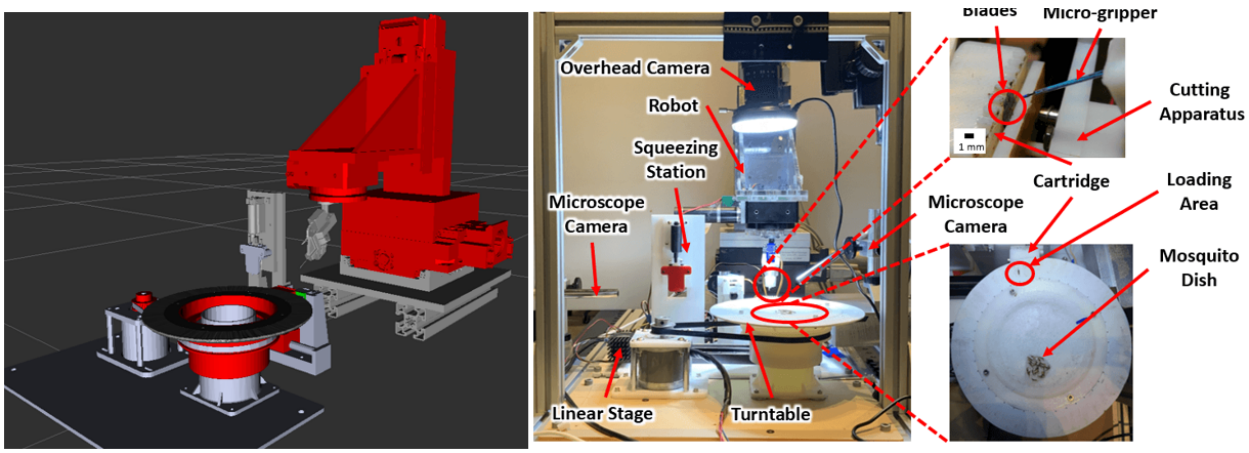

Figure 4: *The simulator (left) and the physical system (right) for the paper under review for ICRA21*

Additionally, the ICRA 2021 paper introduced a simulator (fig 4) for the physical system which is planned to interface with the control algorithm. Using the simulator, we are able to specify the error rate and safely & remotely test the robot control algorithm.

Since our submission to ICRA 2021, our team has made significant progress towards a new concept that aims to execute steps in the micro-dissection procedure in parallel. In the new concept we use independent subsystems, referred to in this report as "nodes". As seen in figure 5b which describes our new concept, the robot, gripper, and cutter are grouped into a *Mosquito Pick-Place-Decapitate* (MPPD) node. The squeezer node which expels and collects the gland is renamed as the *Gland Extraction Node* or simply the *Extraction Node*. Next, we added a *Cleaning Node* which removes mosquito bodies after the gland is extracted. Unlike the ICRA 2021 paper, mosquitoes move between nodes using the *Turntable Node* which was updated to include slots to hold the mosquitoes (see figure 6).

6 Goals

The goal for my Computer Integrated Surgery II project was to develop a next generation control algorithm for the new concept (fig. 5b and 6) that includes parallel processes, error checking, and error recovery. We defined each idea as follows:

- **Parallel Processes** The final delivered controller must be able to support the operation of several steps simultaneously. For example, the controller should be able to operate the

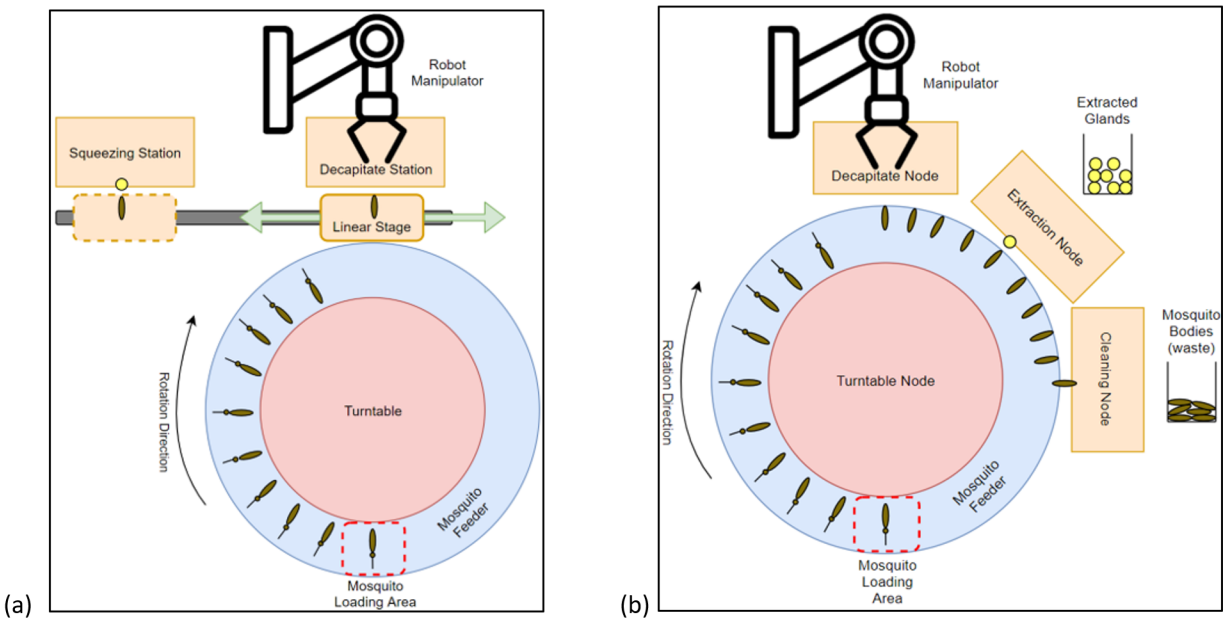

Figure 5: *(a) A symbolic diagram showing the contents and placement of our* ***old*** *robot system which was published in our ICRA 2021 paper submission. (b) A symbolic diagram showing the contents and placement of our* ***new*** *system which consists of independent nodes surrounding a turntable node.*

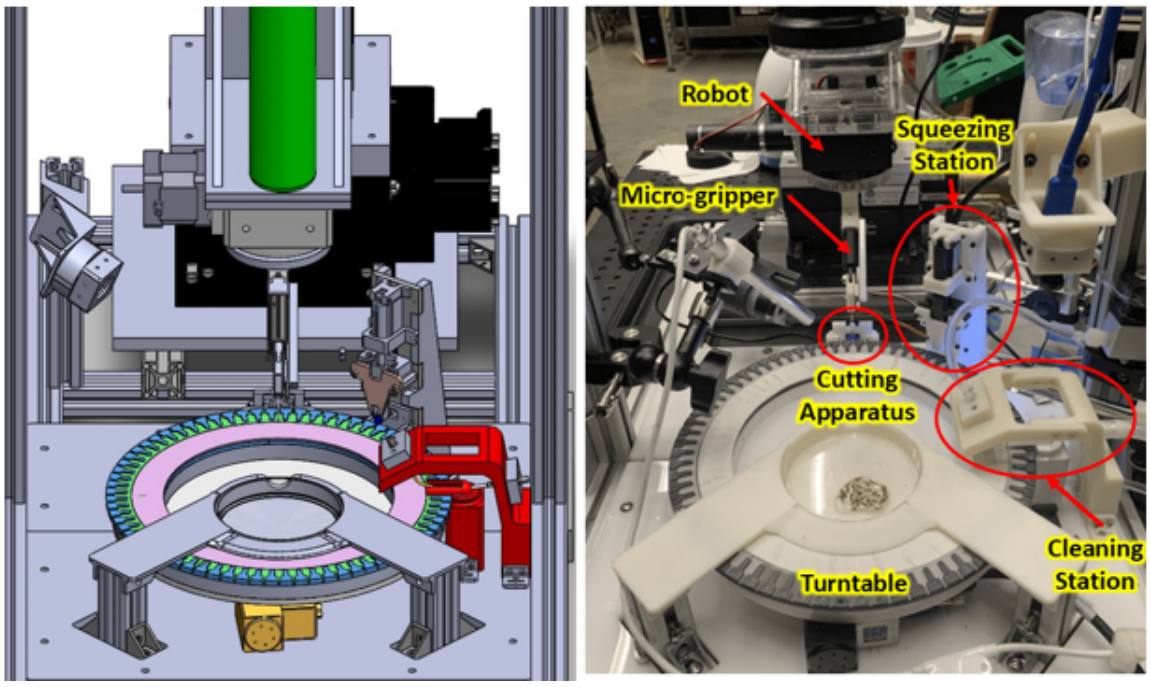

Figure 6: *Next generation hardware system. My project was focused on this hardware. The left image shows the virtual CAD model, and the right image shows the physical system. Subsystem 1 contains the robot, micro-gripper, cutting apparatus. Subsystem 2 contains the squeezing station. Subsystem 3 contains the cleaning station. Subsystem 4 contains the*

robot manipulator while squeezing and extracting the glands of other mosquitoes. The purpose of parallelization is to increase the efficiency of the workflow over the sequential workflow proposed in earlier prototypes.

- **Error Checking** The automated system must be able to recognize as many errors as possible. Potential errors include mosquito status errors (pertaining to specific mosquitoes in the workflow), slot status (such as "empty slots"), hardware errors (such as unresponsive components), and vision errors (such as failures returned by the computer vision algorithm). The purpose of error checking is to maintain safety, prevent collisions, and ensure a high quality in the extracted glands.

- **Error Recovery** The controller must implement logic to resolve errors where possible. Within the scope of the project, the controller should be able to recover from mosquito status errors such as an unsuccessful grip, invalid placement within slots, and invalid extraction.

In addition to developing a robust and successful controller, I was also responsible for several other tasks. As I am not alone on the team, I am responsible for supporting developments in the hardware and computer vision by giving team-wide updates on my weekly progress and discussing plans for integration. After an initial controller is created, I was expected to conduct testing on the physical system and the simulation. I was also responsible for creating detailed documentation for my code which will be handed off to future researchers.

It is important to note the degree to which my project was dependent on the work of others on the team. The controller's level of success depended on the maturity of the hardware and vision components. Dependency details are shown in figure 9 and will be discussed later. Overall, I met the deliverables shown in figure 7, where the green blocks represent completed tasks, yellow blocks represent in-progress tasks, and red blocks represent tasks that were not attempted. I plan to finish the yellow tasks during the last weeks of my Masters program.

7 Timeline & Dependencies

The timeline for the project (fig 8) shows weekly achievements and delivery dates for the deliverables specified in fig 7. We reached our minimum deliverable on April 24th, 2021 and achieved our maximum deliverable for testing on May 1st, 2021.

Because my work is integrated with the hardware and computer vision teams, there are also several dependencies we considered. Figure 9 summarizes the dependencies, the need justification, resolution deadline, final status, and adjustments made. Because of the hardware team was not yet ready to deliver the robot cleaner and hardware, we modified our original minimum deliverable of five nodes (MPPD, Extract, Turntable, Robot Cleaner, Turntable Cleaner) to three nodes (MPPD, Extract, Turntable). Additionally, because the CV team delivered later than expected, I simply put placeholders (i.e. simply pause 3 seconds, then return success) in place of many CV components. The simulation was also delivered late by another team member because he was writing a paper. Thus, we adjusted our timeline to perform simulation testing concurrently with the physical testing, much later than planned.

	a. Minimum Deliverable	b. Expected Deliverable	c. Maximum Deliverable
1. The Basics	Fully debugged, well documented controller; Create ~~five~~ three independent nodes to call services to execute flowchart steps.	Create advanced figures and documentation in the README files and wiki.	Future-proof the system by writing highly generalizable and inheritable code. Create abstract objects, etc.
2. Error & Recovery	Implement the errors & recovery shown in the flowchart. Define / document other errors.		Work with CV team to implement further error handling; one example is dragging detection. Another is ensuring neck is between blades.
3. Integration	Controller operates on the new hardware / setup. Focusing on using placeholders instead of Low-Level Arduino/Galil control.	Write Arduino/Galil code for low level control of hardware. Create working services for the physical robot.	
4. Testing	Visual testing using the simulation or hardware. Component Testing.	Testing on select hardware on the physical system.	Quantitative testing on the physical system to determine success rate.

Figure 7: *The deliverables for this project by category and deliverable level (as of 5/7/2021). Green blocks represent completed items. Yellow blocks represent items that were started, but cannot be considered finished. Red blocks were not attempted.*

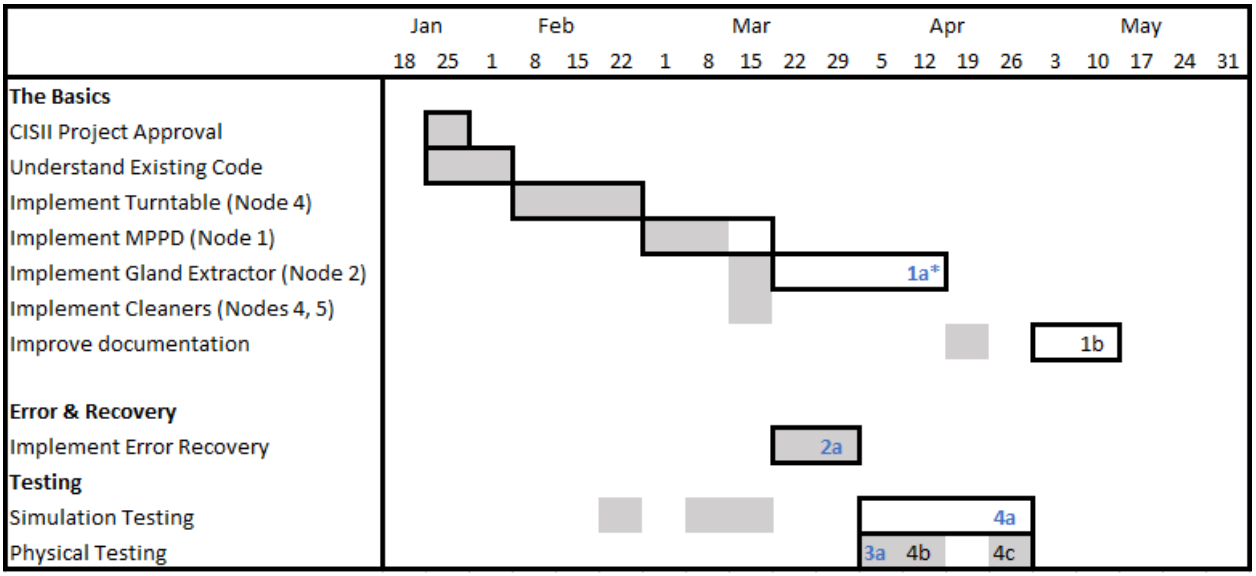

Figure 8: *Development timeline with deliverable deadlines that correspond to definitions in figure 7. The text in blue represent the deliverables needed to meet the minimum product. The shaded regions represent the original planned schedule, and the outlines represent the true development schedule.*

Dependency	Need	Deadline	Status	Explanation	Recovery / Adjustment
Swipe Access to Robot in Lab	Testing and Debugging during development	2/1/2021	Resolved	Obtained	N/A
Error CNN from Computer Vision team	Used for initiating some recovery paths in flowchart	3/22/2021	Partially Resolved	Most MPPD CV was developed, others were developed late / not ready for integration	Put placeholders in the code for this step.
Server/Client Structure is Insufficient	We use this structure to ensure robust communication	3/22/2021	Resolved	Able to adapt architecture for our use.	N/A
JHU Remains in semi-open state	Needed to conduct testing on the physical hardware	4/20/2021	Resolved	JHU lab space remained accessible throughout the semester	N/A
Simulator needs to be developed	The simulator is an important bridge to hardware testing	3/15/2021	Resolved (Late 4/10/2021)	Due to a paper submission, the lab did completed this late.	Simulation testing occurred with the physical testing.
Finished collector hardware	Needed in order to fully develop the gland extract node.	3/29/2021	Partially Resolved	The collector microcontroller code was not finished by other teammates	Put placeholders in the code for this step.
Finished robot cleaner hardware	Needed in order to fully develop the MPPD node.	3/15/2021	Not Resolved	Robot cleaner has not been built. Will likely be built next year.	Robot cleaner node was not attempted
Finished turntable cleaner hardware	Needed in order to fully develop the turntable cleaner node.	3/15/2021	Resolved (Late 4/25/2021)	The microcontroller code was developed late	Turntable cleaner node was not attempted, ran out of time to add placeholders.

Figure 9: *Dependencies, the need statement, final status, and adjustments made to accommodate the final status.*

8 Technologies & Background

We are using an open-source framework called Robot Operating System (ROS) to implement our robot controller. ROS is a collection of tools and libraries that simplify the complex task of robot control. [1] At the core of ROS is the node, a logical unit that can communicate with other nodes using ROS messages and execute scripts to move the robot, call a neural network, output to a GUI, and more. Communication between nodes occurs on ROS topics.

In order to structure the code to allow for error recovery and ease of communication, we are employing a ROS package known as *Actionlib*. Actionlib's *ActionClient* and *ActionServer* communicate via a "ROS Action Protocol" which adds more reciprocity between nodes compared to vanilla ROS messages. The client is the caller which requests the server, the callee, to execute a task and return the result. Thus, the client-server architecture provides the basic functions to efficiently operate the robot system (fig 10).[2]

Communication between an action server and an action client occur on a namespace with four ROS topics: goal, status, feedback, and result. A *namespace* can be thought of as a prefix used in the naming of ROS nodes and topics. For example, according to figure 11, communication between the approach client and approach server occurs on the "/approach_server" namespace. Thus, the four topics used by the pair are "/approach_server/goal", "/approach_server/status", "/approach_server/feedback", and "/approach_server/result". The goal topic is used to send a goal messages from the client to the server. Upon receiving a goal message, the server will send a "goal accepted" status update back to the client through the status topic. The status topic therefore helps ensure the server is online and correctly connected. After the server executes the goal, a message is returned on the result topic which signals to the client that the goal is done. The result message can be unpacked to determine whether the executed goal was successful or encountered an error. Lastly, the feedback topic is used to communicate additional information between the client-server pair. [2]

Our lab has already implemented a similar controller without error recovery for our last hardware iteration using the client-server architecture. My project will focus on supporting the new hardware, adding error recovery, and parallelizing processes. Our new controller will

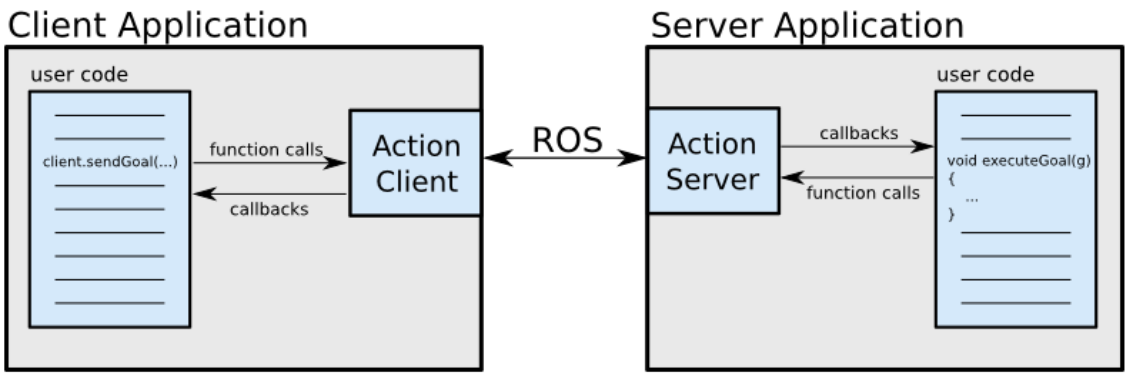

Figure 10: *Two application communicating through an ActionServer and ActionClient pair.*

be based on a flow chart (fig 20) which has five parallel nodes which each control independent subsystems: the Mosquito Pick-Place-Decapitate robot (MPPD), gland extractor, turntable cleaner, turntable, and robot cleaner. The use of the ActionServer and ActionClient architecture can be see in figure 11.

Our lab has two means for debugging and testing. The first option is testing on the physical robot which is located in JHU Laboratory for Computational Sensing and Robotics (LCSR) robotorium and can be accessed and operated remotely as well. The second option is an ROS RVIZ simulation which I helped develop for the last hardware iteration. It needs to be updated to the newest hardware in order to be used for the purposes of this CIS II project.

9 Controller Developments

Over the course of the project, I implemented three independent controller nodes and conducted rigorous testing on the implemented nodes. My efforts resulted in the creation of new logic, the addition of error recovery, and the transition to the new hardware system. I also created the flowcharts, figures, and documentation to aid in explaining my work.

When I first started the project, many of the high level client-server pairs and most low level client-server pairs were fully implemented for the previous hardware system developed for the ICRA 2021 paper submission by W. Li et al. Although the new hardware transitioned from a sequential process to a parallel one, most existing client-server controller pairs did not need very much modification to support the parallel process. Although error recovery was not implemented when I joined, the necessary framework for the idea was present, resulting in smooth and fast development of error recovery.

9.1 Controller Architecture

Please note: Most of section 9.1 was already developed in the work on the ICRA 2021 submission. I made changes to adapted the controller to the new system and add error recovery. Sections 9.2, 9.3, and 9.4 describe my specific work in detail.

9.1.1 Action Client & Action Server Usage in Architecture

The controller is built as a hierarchy that executes actions with decreasing scope moving down the hierarchy. At the top is the *node*, which acts as the central concept of the parallel system design. Nodes are mutually independent and capable of making their own decisions. All nodes communicate with the turntable which acts as a unifying step because the turntable can only actuate when all other nodes are ready. Each node is comprised of *high-level clients* and some connecting logic. As an example, the MPPD node contains an Approach Client, Pick Client, Place Client, and Decapitate Client. Each of these clients contains high level logic on a portion of the MPPD process. Error recovery also occurs in these clients because they make recovery decisions based on results returned from the high level servers. If several attempts at error recovery fails, the high level clients will notify the MPPD node which can choose to give up on the current mosquito by calling to advance the turntable or self-shutdown to prevent dangerous failure. Communication between high-level pairs occur on specific topics specified in figure 11.

High-level servers also contain low-level clients which send requests to low-level hardware and vision servers. As in figure 11, the approach server contains three low-level clients: robot move client, CV mosquito finder, and CV proboscis finder. When the approach server requires a robot movement, it uses the low-level robot move client to sends a goal to the low-level robot move server using the "/sanaria/robot/goal" topic. The robot move server interacts with the Galil DMC-4143 micro-controller to actuate the robot. If an error occurs, the robot move server reports to the robot move client. Depending on the severity and impact of the error, the recovery may occur within the robot move client or within the high-level clients.

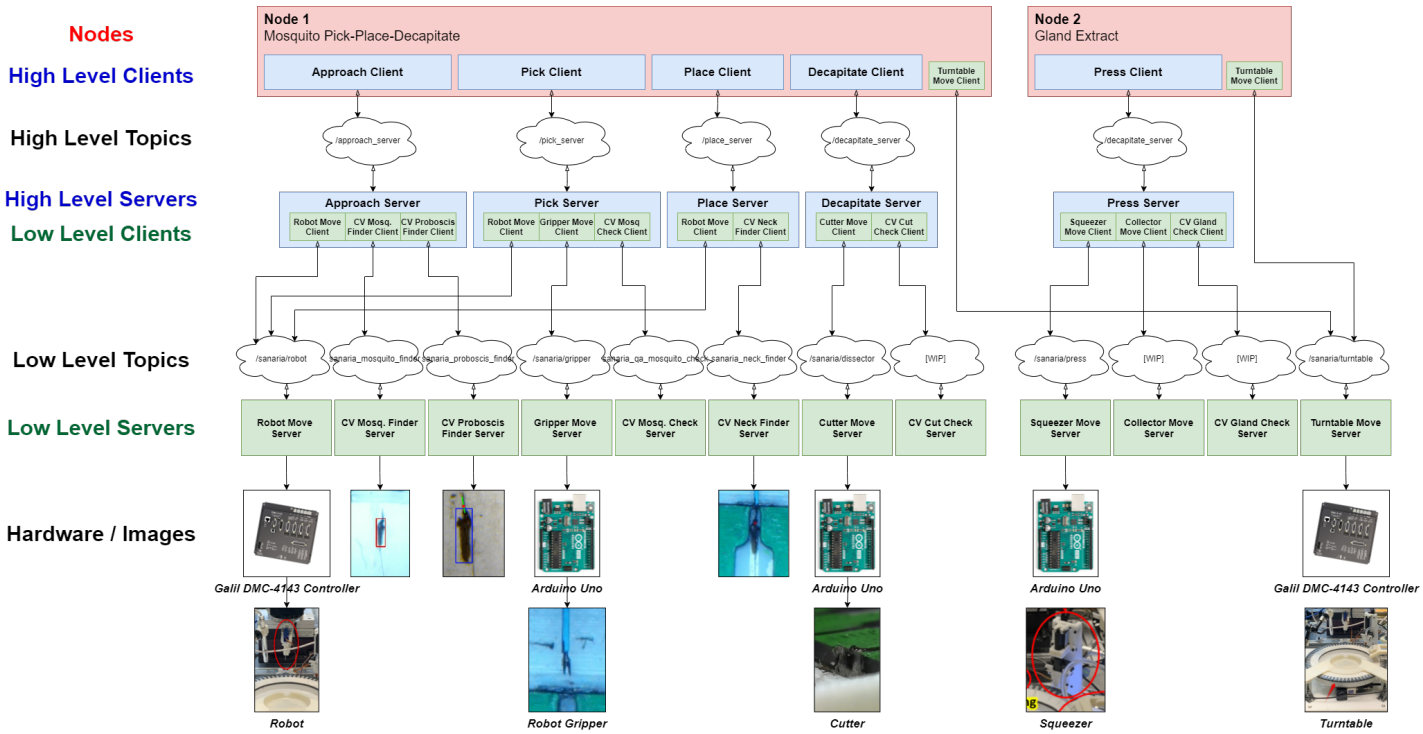

Figure 11: *Full map detailing the action client/server architecture for 2 nodes: Mosquito Pick-Place-Decapitate (MPPD) and gland extractor. Matching blue and green objects are action client/server pairs. They communicate via a topic shown with a "cloud object". For an interactive diagram visit: Sanaria Client Server Architecture*

9.1.2 Error Definition Through Actions

Errors are defined and organized according to the ROS ActionLib package using the **.action* type files. Action files allow users to define custom messages for the goal, feedback, and result topics. The messages can convey integer variables, floats variables, string variables, array variables, constants, and other ROS messages between client-server pairs. [2] Each topic in figure 11 is defined by a unique action file. For example, the approach client-server pair communicates using the messages defined in the *Approach.action* file. The definition for *Approach.action* is shown in listing 1.

```
# approach action for Sanaria actions
# this action should send the gripper to the mosquito proboscis, actuate the
    gripper, and secure the mosquito in the gripper

#goal definition
sanaria_msgs/ApproachGoal goal
sanaria_action/BaseGoal base_goal

---
#result definition
sanaria_msgs/ApproachResult result
sanaria_action/BaseResult base_result

---
#feedback definition
sanaria_msgs/ApproachFeedback feedback
sanaria_action/BaseFeedback base_feedback
```

Listing 1: *Approach.action* Each definition consists of two ROS messages a Approach-specific message and a Base message

Furthermore, we see that each definition in listing 1 contains an ApproachX and a BaseX ROS message. Here X represents the sensible choice from [Goal,Result,Feedback]. ApproachX defines messages specific to the client-server pair, and BaseX defines messages that are general such as a hardware failure or power outage. As an example, ApproachGoal messages contain information specific to the current goal. Listing 2, contains the possible commands which can be sent to the Approach Client including actions such as NORMAL, DETECT, and VALIDATE.

```
# Approach goal
uint8 goal_code

# constants
uint8 NORMAL = 0        # normal operation from start to finish
uint8 DETECT = 1        # detect mosquito on the turntable
uint8 ROBOT_MOVE = 2    # move robot to the target position
uint8 VALIDATE = 3      # validate that the robot is at the correct position
```

Listing 2: *ApproachGoal.msg* The goal consists of a goal_code command variable and constant definitions of possible goal codes.

```
# Approach result
uint8 result_code

# constants
```

```
uint8 SUCCESS = 0            # approach goal was successfully finished
uint8 FAILED_DETECT = 1      # CV Mosq. Detect unable to find mosq.
uint8 FAILED_ROBOT_MOVE = 2 # Robot unable to move
uint8 FAILED_VALIDATE = 3    # CV failed to validate approach
```

Listing 3: *ApproachResult.msg* The result consists of a result_code variable and constant definitions of possible result codes.

Upon finishing a goal, the Approach Server returns listing 3 to the Approach Client. Based on the error type found in the result_code, the client may attempt error recovery or give up and move to the next mosquito.

9.2 Turntable Node

The turntable node is responsible for safely rotating the table once it has received an "increment turntable" goal from each of the other nodes. Rotating before a node is ready may cause physical damage to hardware components if the squeezer or gripper tips are still within a slot. It should be noted that the turntable node is not a true node as defined in section 9.1.1 because it is actually a low-level server. It may be more accurate to refer to the turntable node as a service. The strength of this architecture is its ability to avoid unnecessary complexity. If the turntable was programmed as the earlier defined "node", we would need to create a high-level (client-server) pair and a node to wrap around this pair. We argue this structure adds unnecessary complexity to the program because the commands and errors of the turntable are relatively simple compared to with other nodes that require a combination of hardware and computer vision. However, this design has drawbacks; notably, a lack of uniformity may make future development more difficult and confusing. However, we reasoned it would be simple to convert our turntable into a true node.

The logic of the turntable, handled in the low-level server, is seen in figure 12. To accomplish this task, we used a std::vector to store pointers to each TurntableMoveGoal that was received from other nodes. The goals have unique caller identifiers that allow the turntable to ignore repeated requests from the same node. This also prevents one node from activating the turntable by simply sending three "increment turntable" goals. After the turntable executes, it returns one result message on the "/sanaria/turntable/result" topic for each node that sent a goal.

Although seemingly simple, designing the turntable node that can queue incoming goals required diving into the implementation of the ActionServer and ActionClient since traditional client-server pairs will abort a previous goal as soon as it receives a new goal.

The turntable client interacts with the hardware with an unused fourth axis on the Galil DMC-4143 micro-controller which also powers the 3-axis robot.

9.2.1 Slot Statuses

Slot statuses were needed to track the state of mosquitoes within each slot. Slots with mosquitoes that were successfully decapitated by the MPPD node should be operated on by gland extraction and turntable cleaning nodes. On the other hand, empty slots or slots with unsuccessful mosquitoes, should be marked to prevent the gland extraction node from operating pointlessly when the slot reaches the it. We accomplish slot status tracking using an integer array stored as a ROS Parameter named "slot_statuses". The definitions for

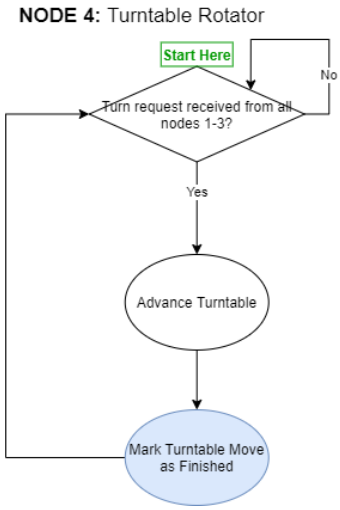

Figure 12: *Flow chart for the turntable node.*

the integers standardized in the TurntableMoveGoal ROS message (listing 4). On startup, the turntable node sets all 72 turntable slots to "UNKNOWN". If the MPPD node successfully decapitates the mosquito, the slot is set to "OK". Otherwise, it is set to "EMPTY" or "FAILED_APPROACH". The gland extraction node checks the status of the slot in front of it and only operates on an "OK" status. The turntable cleaner node will clean any slot that is not marked "EMPTY". Additionally, each time the turntable finishes a "increment slot" request, the turntable's slot number ROS parameter is incremented.

```
...
# Slot Status Options
uint8 OK                = 0
uint8 UNKNOWN           = 1
uint8 EMPTY             = 2
uint8 FAILED_APPROACH   = 3
uint8 FAILED_PICK       = 4
uint8 FAILED_PLACE      = 5
uint8 FAILED_DECAPITATE = 6
uint8 FAILED_SQUEEZE    = 7
uint8 FAILED_EXTRACT    = 8
uint8 FAILED_CLEAN      = 9
...
```

Listing 4: *Excerpt from TurntableMoveGoal.msg containing definitions for slot statuses*

9.3 Mosquito Pick-Place-Decapitate (MPPD) Node

The Mosquito Pick-Place-Decapitate (MPPD) node commands the robot and gripper to place the mosquito between the cutter and to decapitate the mosquito head. The MPPD process is broken into four sub-process which occur sequentially: Approach, Pick, Place, and Decapitate. Each process is identified in figure 20 using a color background block and implemented using four distinct high-level client-server pairs in the architecture map in figure 11. Specifically, each sub-process is described in detail below

- **The Approach Sub-Process** is responsible for rotating the turntable until a mosquito

is detected on the turntable. Then, the approach process will us the CV proboscis finder to determine pixel coordinates to position the gripper. The controller converts the pixel coordinates to a robot coordinates in units of encoder counts and moves the robot above the robot coordinates. The process flowchart is show in figure 13.

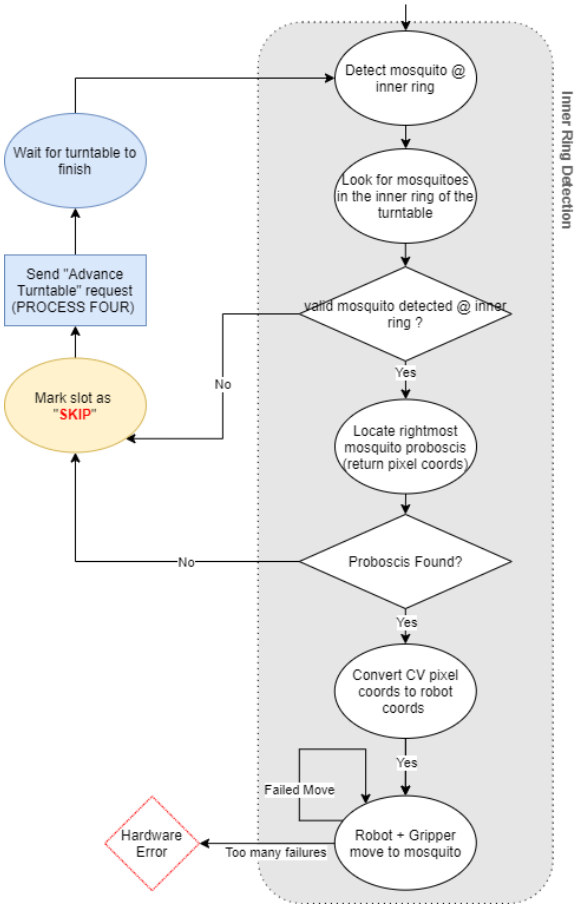

Figure 13: *Flow chart for the approach sub-process, the first sub-process in the MPPD node.*

- **The Pick Sub-Process** is responsible for securely gripping the mosquito proboscis. It starts with the gripper above the proboscis and lower the robot until the gripper contacts the table and the proboscis. It then closes the gripper and attempts to drag the mosquito to a different position closer to the cutter. If the CV determines that the mosquito did not move from the original position on the inner turntable, then the algorithm will attempt a "re-pick" recovery. The process flowchart is show in figure 14.

- **The Place Sub-Process** is responsible for placing the mosquito with the neck between the blades and the body within the slot. The robot will drag the mosquito from the validate pick position to a position within the slot but before the cutter. The CV algorithm will calculate the neck position in pixel coordinates which is then used to ensure the robot can place the mosquito with its neck between and aligned with the blades. If the CV algorithm is unable to recognize the neck, a small adjustment is performed within the slot so we can get a different body orientation to attempt to detect the neck again. The process flowchart is show in figure 15.

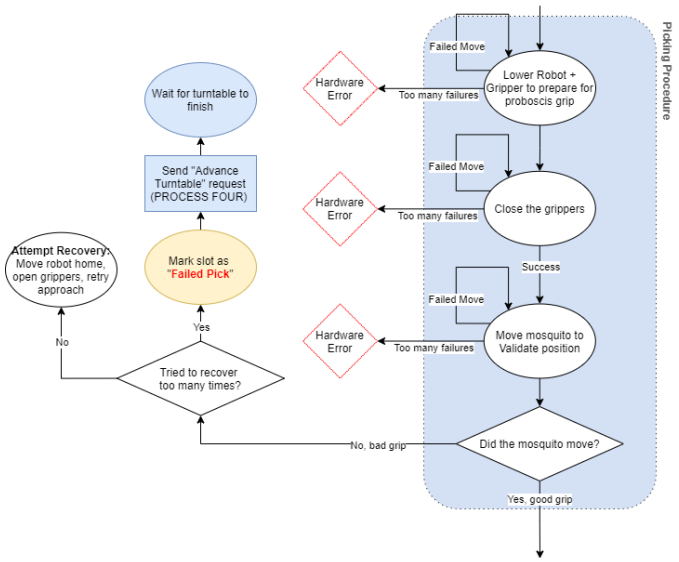

Figure 14: *Flow chart for the pick sub-process, the second sub-process in the MPPD node.*

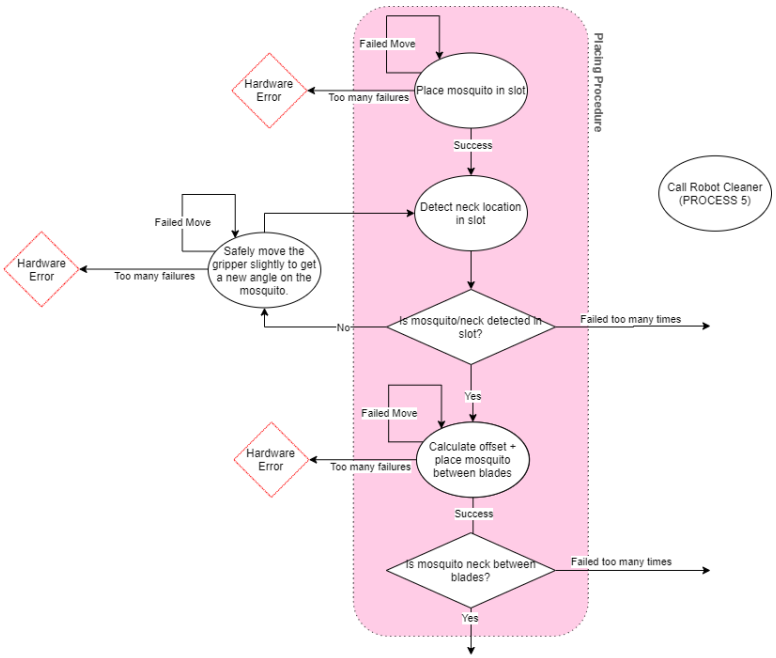

Figure 15: *Flow chart for the place sub-process, the third sub-process in the MPPD node.*

- **The Decapitate Sub-Process** is responsible for separating the mosquito head from its body. The process begins by actuating the cutter. Then, with the proboscis still held with the gripper, the robot moves away from the blades to separate the head. Although our current CV algorithms do not allow use to recognize if the head is successfully detached from the body, I created a place holder in the decapitate sub-process could validate the step in the future and attempt error recovery. The process flowchart is shwo in figure 16.

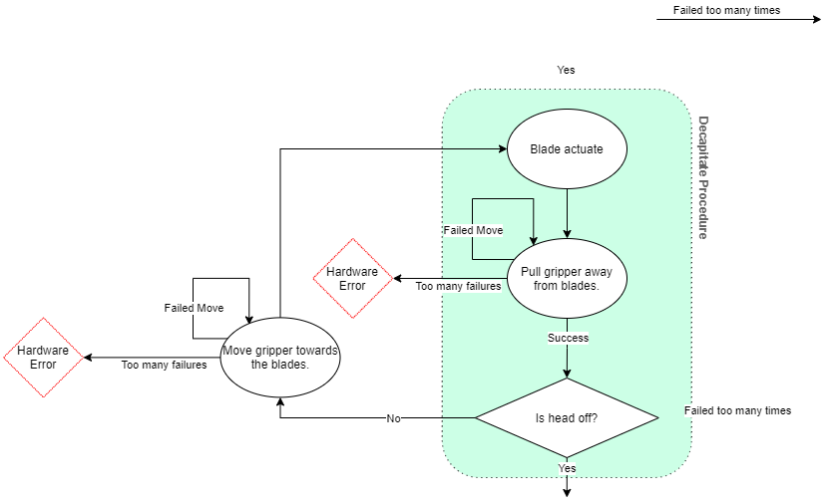

Figure 16: *Flow chart for the decapitate sub-process, the fourth sub-process in the MPPD node.*

The MPPD node greatly relies on the low-level robot node which recieves goals from the Approach, Pick, and Place sub-processes as shown in figure 11. The robot node is connected to Galil DMC-4143 hardware controller through a Ethernet cable. The motor PID values and the encoder positions for each validation step were carefully tuned by Wanze Li and me. These tuned values contribute largely to the success rate of the system.

9.4 Gland Extraction Node

Located downstream of the MPPD node, the gland extraction node is responsible for squeezing the mosquito thorax to extrude the salivary gland from the decapitated neck. A second blade is used to separate extruded gland, and a gust of water will wash the gland into a receptacle. This step is directly responsible for collecting the salivary glands, the output of the process, so it is critical to use computer vision to recover errors and problems with extracted gland quality. Gland extraction follows the flowchart in figure 17. The node begins by checking the "slot_statuses" ROS parameter for the status of the current node. If the slot status is empty or erroneous, the node sends a turntable "increment slot" and awaits the next node. If the slot status is "OK", the squeezer is lowered to apply pressure to the mosquito throax. A CV algorithm, which was not ready at the time of this project, will check the mosquito neck area for extrudate. If detected, the collection process occurs to separate and obtain the gland; otherwise, the controller raises and lowers the squeezer again as a recovery step. After collection, the CV algorithm is called to detect the salivary gland again. If the gland is no longer there, we assume the gland has been obtained. Otherwise, we attempt error recovery by attempting to collect again. After collection validation passes, we raise the squeezer and request that the turntable "increment a slot".

Currently the collector and gland detection CV algorithm low-level nodes are not implemented because the hardware and CV components are not ready. Thus, I put placeholders in the code that always return "success" results at these location. When ready, it should be simple to add these components.

Figure 11 displays the architecture of the gland extractor node. The node consists of a single high-level press client-server pair. The press server contains the low-level hardware and validation clients which are used to complete the gland extraction process described earlier in this section.

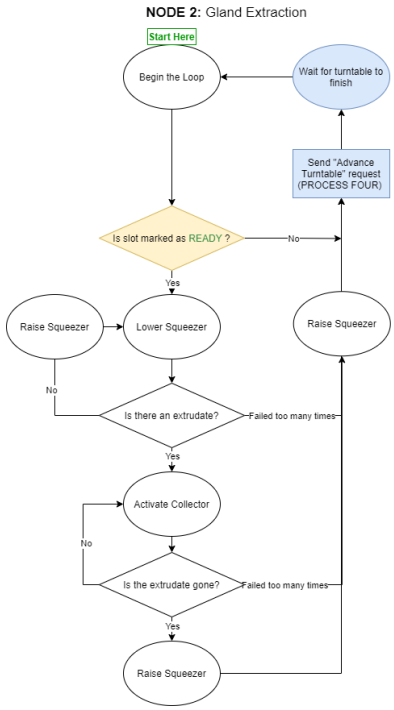

Figure 17: *Flow chart for gland extraction node which is used to squeeze and collect the extruded gland from a decapitated mosquito. Although not marked, each squeezer move is accompanied by a check for hardware error similar to those described in the MPPD processes.*

9.5 Integration With Simulator

With the help of Wanze Li, I created a simulator for the old hardware system during the summer of 2020. During the CIS II project, I helped update and test the simulation of our new system. The simulation was created in the ROS Robot Visualization tool (RVIZ), a light-weight tool which easily connects with ROS nodes to help virtually visualize the position of robot meshes. By subscribing to the goal topics of the low-level client-server pairs, the simulation was able to run independently of the physical system or mirror the motions of the physical system (as shown in fig 18). This gave us the opportunity to test out or controller in our virtual system first. We found several collision-causing errors through the simulation that would be damaging on the real hardware.

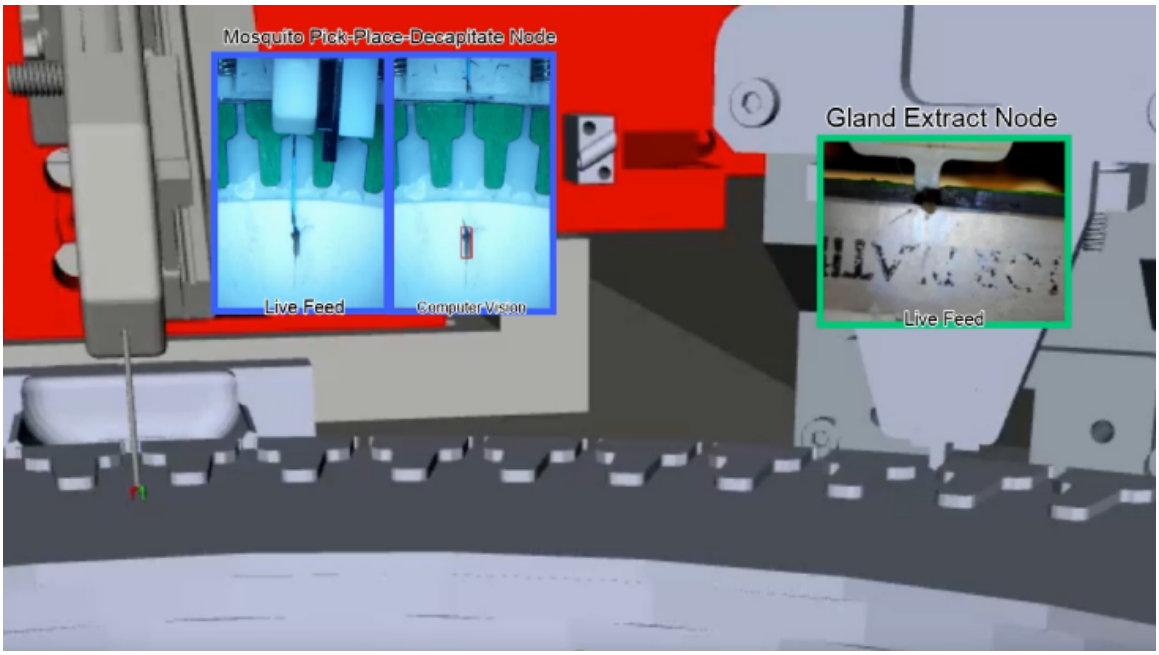

Figure 18: *Synchronous images of the system and the state of the simulation confirm that the simulation is able to mirror system state. Notice the lowered squeezer is mirrored in the simulation as well as the position of the robot and gripper.*

10 Testing

Our testing plan was composed of three levels: component testing, system testing through the simulator, and system testing on the hardware. Each level introduced more complexity and our passing of all three resulted in a successful deliverable.

10.1 Level 1: Component Testing

Component testing occurred during the development of each node. I wrote a script to command the node to execute both simple and edge actions to attempt to break the node. I made improvements to the node if any issues arose. I detected potential issues by monitoring the topics used by client-server pairs for abnormalities. In cases where collision is imminent, component testing occurred on the simulator before the hardware. Figure 19 shows the monitored output of the turntable node as it underwent component testing on the physical system.

10.2 Level 2: System Testing on Simulator

System testing on the simulator consisted of creating scenarios and generating a combination of errors on the simulator. The scenarios were created using a live camera stream on the real system with different positions of mosquitoes and slots. The physical components were turned off as I watched the controller attempt a variety procedures on the simulator to look for abnormalities and bugs.

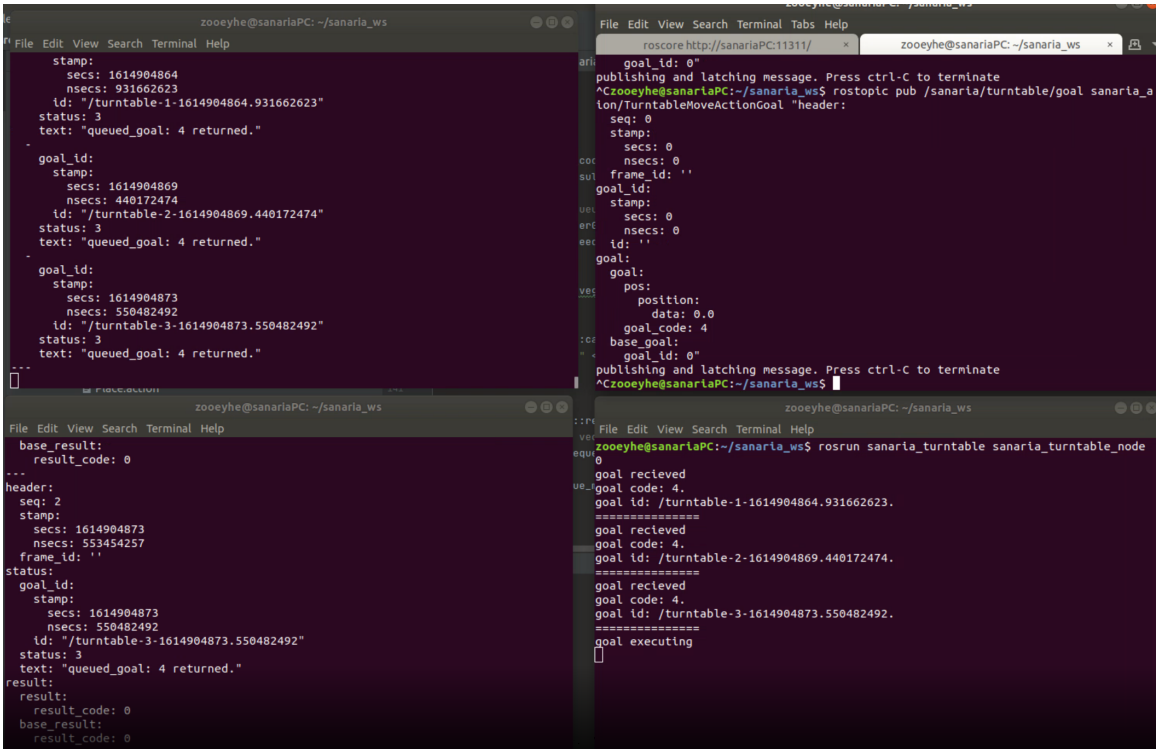

Figure 19: *Level 1 component testing on the physical turntable hardware. The four terminal windows monitored the output of the low-level turntable node. From top-left to bottom-right, the terminals monitored "/sanaria/turntable/status", "/sanaria/turntable/goal", "/sanaria/-turntable/result", and the direct output of the node.*

10.3 Level 3: System Testing on Hardware

System testing on hardware accomplished the same error recovery testing as level 2 and helped determine the mosquito throughput and success rate measured in mosquitoes per hour (mph). We can compare the mph to previous controllers and systems to determine whether the parallel controller with error recovery contributes to a rate gain. Level 3 testing depended on developments from the hardware and computer vision teams (see fig 9) as throughput and success rates depend on the effectiveness of hardware and computer vision models.

11 Results & Discussion

As part of the deliverables, I produced several videos of the physical and simulated system working. There are hours of experimentation video, however the selections below are good examples to demonstrate error recovery and parallelism.

- A video of the MPPD pick error recovery: https://youtu.be/fm2GvXleZxQ
- A video of the MPPD and Gland Extraction nodes parallelism: https://youtu.be/3QZfyzyuBGo

Additionally, I conducted testing on 30 mosquitoes and manually determined the success rate, error recovery rate, and throughput of various components in the system. The summary data is displayed in table 1 which shows that the number of total successes (initial success count + recovery count), the step success rate, and the cumulative success rate. The step success rate is calculated using $\frac{step\_success\_count}{success\_count\_of\_previous\_step}$ while the cumulative success rate is calculated using $\frac{step\_success\_count}{total\_mosquitoes\_at\_start}$.

Node	Step	Success Count	Recovery Count	Total Success Count	Step Success (%)	Cum. Success (%)
N/A	Start	30	—	30	—	—
MPPD	Approach	30	0	30	100%	100%
MPPD	Pick	26	4	30	100%	100%
MPPD	Place	24	0	24	80%	80%
MPPD	Decapitate	16	0	16	67%	53%
Gland Extract	Squeeze	12	0	12	75%	40%

Table 1: Raw data determining the number of success at each point in the automated system.

Table 1 demonstrates several key ideas and observations. The cumulative success rate for the MPPD node is only 53% largely because of the decapitate step with a individual success rate of only 67%. In our observation, the decapitate success rate could be improved through better placing of the mosquitoes between the blades. Often, mosquito bodies were above the blade's height. When the blades actuated, these mosquitoes do not become successfully decapitated; their bodies were simply pushed up during actuation which avoids the cut. The problem could be avoided with taller blades and the implementation of the decapitate validation step. Additional hardware to keep the mosquito bodies within the slots could also mitigate the issue.

Due to a lack of mature hardware and CV, we were only able to perform quantitative testing on error recovery for the pick and place step. However, error recovery allowed us to save 100% of missed mosquitoes by the the pick step, demonstrating the potential of the CV error detection and controller recovery.

Using a handheld timer, we determined the average time for the MPPD node to process one mosquito to be roughly 10s. The MPPD node operates for longer than the gland extraction node, thus we can use the MPPD node to estimate the mosquito throughput at 360 mosquitoes per hour (mph). The gripper actuation uses a slow linear servo, which could easily be replaced to remove around 2s per mosquito, improving the rate to 450mph. Since our goal rate is 600mph, we could investigate increasing the robot move speed and cutter speed to reach our goal.

11.0.1 Lessons Learned

The simulator was helpful to us during testing and debugging of the control algorithm. It served as a safe and effective check for logic errors in our components and our system synergy. Currently, we still need to be present in the lab to run the simulator. In the future, the simulator could be run from any computer and would be a greater development tool for researchers in our lab.

The new controller makes it easier for the hardware and vision team to perform large scale testing on mosquito samples. We were conducting testing using an old controller that did not use ROS and required manual commands for each sub process and step. A tedious procedure, the previous testing required roughly an hour to run experiments on 30 mosquitoes. Currently, we only need roughly 35 minutes on our the new controller.

Personally, I gained very much experience using the ROS ActionServer and ActionClient paradigm. For the turntable node, I performed a deep dive into the source code and learned about the implementation. I also recognized the importance of hierarchical structure to simplify and organize complex controllers.

11.0.2 Future Work

Future work for this project focuses on bringing the the system to maturity. With respect to the controller, our team would like to implement a better start-up calibration procedure, a method to track mosquito statuses in addition to slot statuses, and create a method for recording, documenting, and labelling all past runs to perform failure analysis. Continued support for hardware and vision developments and the implementation of more error detection and recovery would further improve the controller's efficacy. I am currently working on improving the documentation and the repository wiki to note our attempts, rationale, and engineering process for future students on this project.

12 Management Plan

12.1 Team Members and Roles

This team consists of:

- **Zhuohong (Zooey) He** (zhe17@jhu.edu)
  MSE Robotics, Laboratory for Computational Sensing & Robotics, second year
  Sole responsibility for all tasks required for this project.

The mentors consist of:

- **Dr. Simon Leonard** (sleonard@jhu.edu)
  Assistant Research Professor (Computer Science)
  Dr. Leonard is very experienced in ROS development and will help advise on the software architecture design, error recovery logic, and resolving difficult ROS challenges. He will be the primary mentor on this project.

- **Dr. Russell Taylor** (rht@jhu.edu)
  John C. Malone Professor (Computer Science)
  Dr. Taylor has a decorated background in medical robotics and will provide his expertise and leadership to ensure our development is aligned with the team and larger research goal. Dr. Taylor will be a valueable resource and a second mentor on this project.

Our contacts at our industry partner Sanaria include:

- **Dr. Kim Lee Sim**
  Executive Vice President Process Development and Manufacturing, Sanaria Inc.

- **Sumana Chakravarty**
  Sanaria Inc.

12.2 Meeting Schedule and Communication Platforms

Scheduled meetings for this project occurred on a weekly basis and served to update the mentor and the larger research group.

- **Friday 11:30am - 12:00pm:** Update meeting between Zhuohong He and Dr. Simon Leonard. The purpose of this meeting was to discuss technical challenges faced during the week and to coordinate efforts to move according to the timeline.

- **Tuesday and Thursday Evening:** Reserved time to run tests on the physical robot. This time could also be used for extra meetings with Dr. Leonard or Dr. Taylor.

- **Monday 10:00am - 11:00am:** Update meeting with the research group. The purpose of this meeting was to coordinate efforts with the computer vision and hardware teams, to hold each other accountable, and to address concerns.

Communication between Zhuohong He and Dr. Simon Leonard (primary mentor) occurred largely through email. However, efforts were made to exchange phone numbers for faster communication.

13 Reading List

- **First generation mosquito micro-dissection robot introduction** [4]
- **A brief introduction to ROS** [1]
- **A brief tutorial for the Actionlib ROS package** [2]

References

[1] Amanda Dattalo. actionlib ros wiki. http://wiki.ros.org/ROS/Introduction. Accessed: 2021-02-25.

[2] Eitan Marder-Eppstein, Vijay Pradeep, and Mikael Arguedas. actionlib ros wiki. http://wiki.ros.org/actionlib. Accessed: 2021-02-25.

[3] Benjamin Mordmüller, Güzin Surat, Heimo Lagler, Sumana Chakravarty, Andrew S. Ishizuka, Albert Lalremruata, Markus Gmeiner, Joseph J. Campo, Meral Esen, Adam J. Ruben, Jana Held, Carlos Lamsfus Calle, Juliana B. Mengue, Tamirat Gebru, Javier Ibáñez, Mihály Sulyok, Eric R. James, Peter F. Billingsley, K. C. Natasha, Anita Manoj, Tooba Murshedkar, Anusha Gunasekera, Abraham G. Eappen, Tao Li, Richard E. Stafford, Minglin Li, Phil L. Felgner, Robert A. Seder, Thomas L. Richie, B. Kim Lee Sim, Stephen L. Hoffman, and Peter G. Kremsner. Sterile protection against human malaria by chemoattenuated PfSPZ vaccine. *Nature*, 542(7642):445–449, Feb 2017.

[4] Henry Phalen, Prasad Vagdargi, Mariah L. Schrum, Sumana Chakravarty, Amanda Canezin, Michael Pozin, Suat Coemert, Iulian Iordachita, Stephen L. Hoffman, Gregory S. Chirikjian, and Russell H. Taylor. A mosquito pick-and-place system for pfspz-based malaria vaccine production, 2020.

[5] Mariah Schrum, Amanda Canezin, Sumana Chakravarty, Michelle Laskowski, Suat Comert, Yunuscan Sevimli, Gregory S. Chirikjian, Stephen L. Hoffman, and Russell H. Taylor. An efficient production process for extracting salivary glands from mosquitoes, 2019.

[6] Russell H. Taylor, Amanda Canezin, Mariah Schram, Iulian Iordachita, Gregory Chirikjian, Michelle Laskowski, Sumana Chakravarty, and Stephen Hoffman. Mosquito Salivary Gland Extraction Device and Methods of Use, December 2017.

[7] World Health Organization. World malaria report 2019, Dec 2019.

14 Appendix

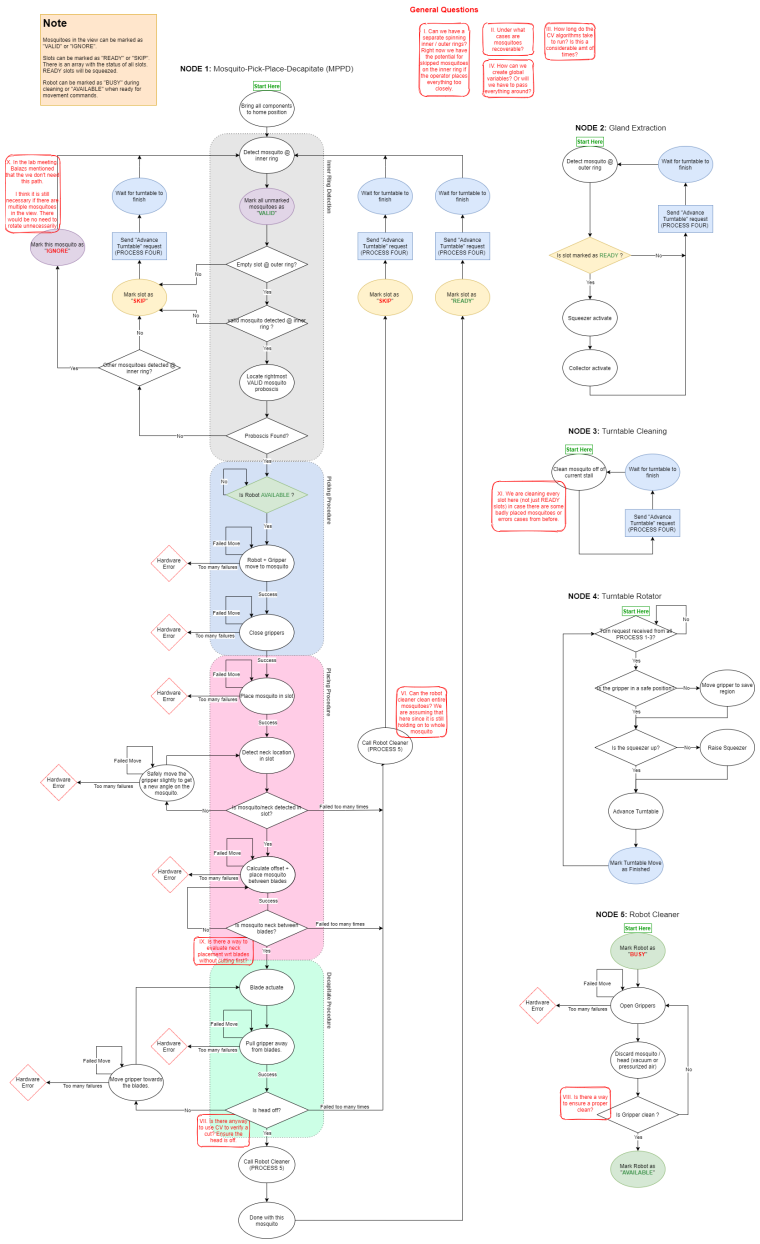

Figure 20: *Full flowchart detailing 5 nodes: Mosquito Pick-Place-Decapitate (MPPD), gland extractor, turntable cleaner, turntable, and robot cleaner. A magnify-able flowchart can be found at: Sanaria Controller Flowchart*