Automation of Mosquito Dissection for Malaria Vaccine Production

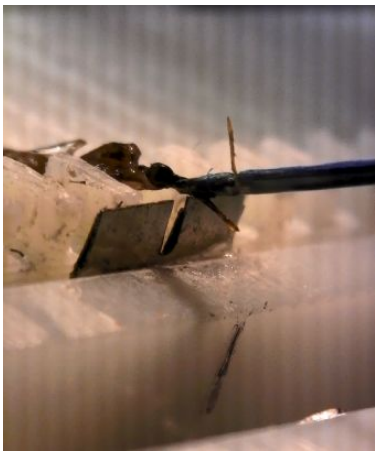
Final Report

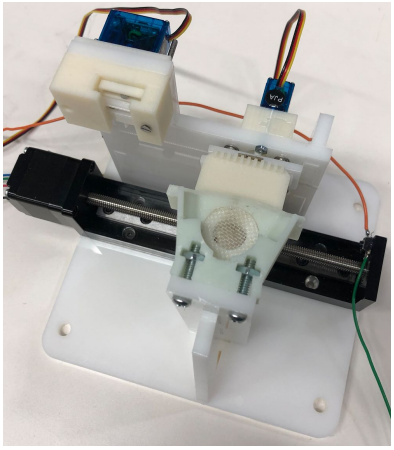
(w/ added software details and walkthroughs by H.P.)

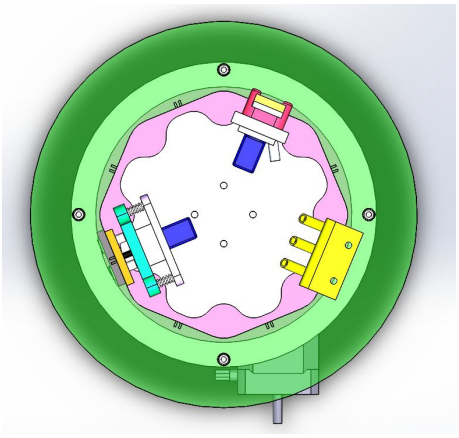
May 31, 2019

Group 1: Henry Phalen, Michael Pozin, Alexander Cohen

Mentors: Dr. Iulian Iordachita, Dr. Russell H. Taylor







LABORATORY FOR Computational Sensing + Robotics
THE JOHNS HOPKINS UNIVERSITY

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Statement of Confidentiality:

This report includes information, designs, and plans for items that have not been publicly disclosed due to intention of pursuing intellectual property. This report and its content, before that public disclosure, is only for the eyes of those who have signed a non-disclosure agreement.

Executive Summary

Sanaria Inc. and the Laboratory for Computational Sensing and Robotics (LCSR) at Johns Hopkins University are collaborating on an NIH Small Business Innovation Research (SBIR) grant to help scale the manufacture of Sanaria's viable Malaria vaccine. Malaria presents an undeniable public health burden to the global community. Annually, over 200 million people suffer from the disease and over 1 million are killed, many of which are children.To help Sanaria address manufacturing issues, researchers at the LCSR have formed a collaborative team to robotically automate Sanaria's production process. The production process utilized currently by Sanaria involves manual dissection of *in vivo* mosquitoes to dissect and collect salivary glands for further processing. The overall goal of this project is to automate mosquito dissection, an expensive, rate-limiting, and training-intensive step in Sanaria's production process, and enable the scaling of the vaccine production.

The scope of this Computer Integrated Surgery II project focuses on the development of two major subsystems within the larger automation effort. In particular, this project addresses the robotic pick-and-place component and the dissection system component directly. For the success of the project as a whole, a collaborative relationship between subsystem developers was maintained between the team members working on this course project and those working on the larger project as a whole.

This report focuses on the development and testing of a robotic pick-and-place system embedded with collaborator-developed computer vision to correctly place a mosquito into the dissection system for subsequent processing. With this system, we demonstrated 100% grasping accuracy and 90% placement accuracy in a test with 50 mosquitoes. The dissection system, which was also addressed under the scope of this course project, attempts to develop a system consisting of several modules that decapitate the mosquito, extrude salivary glands and other exudate from the mosquito, collect the exudate, and dispose of the mosquito body. A linear staging system was developed for the purpose of proving the functionality of the system and a more streamlined approach utilizing a rotary stage was proposed and designed. In addition to subsystem development, a great deal of project effort went into integration exercises, both at the hardware and software interfaces. We intend for this project to continue after the conclusion of our work in the hands of our mentors and other students in the future and have provided substantial documentation of both our final products and decisions along the way for this purpose.

I. Introduction

A. Motivation

Our project's motivation is best summarized with this lightly-edited excerpt from the manuscript our group members submitted to CASE 2019 (Full paper in Appendix B):

Malaria presents a tremendous public health burden. The World Health Organization estimates 219 million individuals worldwide were infected with the disease in 2017 and ranked it among the top 20 leading causes of death among both adults and infants in 2016 [1], [2]. With increasing drug and insecticide resistance, it has become ever more difficult for current treatments to maintain efficacy in reducing the prevalence of malaria worldwide [3]. Development of malarial vaccines present a promising way forward in the global effort for malaria eradication [3]. Progress has been made in the development of the Sanaria *Plasmodium falciparum* sporozoite-based vaccine (Sanaria® PfSPZ Vaccine), an effective vaccine manufactured from PfSPZ extracted from the salivary glands of female *Anopheles* mosquitoes [4]–[9]. Such a vaccine may reduce the burden of the disease by providing immunity against Pf, the most common malarial parasite, which was estimated to account for greater than 95% of deaths caused by malaria in 2017 [1], [10].

The process of vaccine production requires salivary gland dissection and to date has only been demonstrated with training-intensive manual or semi-automated processes, presenting a major bottleneck in the scalability of this vaccine. In traditional manual methods, technicians are presented with freshly-sacrificed mosquitoes and process them one at a time, removing the mosquito's head with a needle under microscope and squeezing out a volume of exudate that includes the PfSPZ-laden salivary glands. The exudate from mosquitoes is collected and processed for the isolation of PfSPZ.

The automation of salivary gland harvesting from *in vivo* mosquitoes has been attempted in the past [11]–[13]. However, no literature supports the success of any such process at this time. A semi-automated mosquito micro-dissection system (sAMMS) has been developed and investigated in prior works within LCSR [14], [15]. In the sAMMS process, a human technician uses micro-forceps to sort mosquitoes into cartridges such that their necks extend between cutter blades. Then, the blades are actuated to cut off all the heads, and a comb-like squeezing device is used to extrude all the exudate, which is collected via a suction device. Early experience has shown that this device roughly doubles the throughput of purely manual dissection and reduces training time to reach peak performance from 39 to 1.5 weeks [15].

While a demonstrable improvement over manual methods, the sAMMS device was developed only as a first step towards a fully automated dissection system to enable large-scale production of enough vaccine for worldwide vaccination efforts. This project attempts to further the success

of the sAAMS device by learning from its successes and incorporating novel contributions to make headway in the development of the fully automated system. In particular, robotic manipulation of the mosquito, the entirety of the dissection apparatus, as well as methods to improve the dissection system efficiency will be explored in detail. Notions of incorporation of the broader project will be made throughout the report, but the focus of this project and report will be on the above mentioned three primary objectives.

B. Scope of Project

To best understand our project in the scope of the ongoing automation system development efforts in the LCSR, we briefly describe the role of our work within the larger effort to develop a fully-automated mosquito dissection system for Sanaria, Inc. being undertaken by several researchers in the Laboratory for Computational Sensing and Robotics (LCSR) at Johns Hopkins University.

We primarily developed a robotic pick-and-place system and automated dissection system that will function as subsystems within the fully-automated mosquito dissection system. This larger system will ultimately take freshly-sacrificed mosquitoes suspended in liquid media and output a collection of mosquito exudate including sporozoite-laden salivary glands. Our concept of this dissection system is provided in Figure 1.

First, a staging apparatus will separate mosquitoes and present them one at a time to the robot. Freshly-sacrificed mosquitoes sit in a basin of solution beneath the system. A spinning rotor in the basin creates a vortex that will carry mosquitoes in solution to the top of a separation cone. This cone has channels in one sector down which water will flow onto a ring of orientable mesh-bottomed turntables. This ring will rotationally index around the cone so that, by controlling the vortex speed and concentration of mosquitoes in the basin, the cups will on average have one mosquito on them once they pass beyond the sector of the cone with channels. At an index beyond the channel, a camera will image a single cup and a computer vision algorithm will determine if a mosquito is present. If so, at the next indexed position, the cup will be rotated to orient the mosquito so that the mosquito's proboscis will point radially outward from the ring. Finally, the ring will be rotated to an index that a sits parallel to a linear stage that will comprise the third subsystem, a dissection assembly line. The development of the staging apparatus is described in detail in [13].

The pick-and-place robot will be positioned on the other side of the linear stage and will reach over to the cup, grasp the mosquito by its proboscis and drag it onto a cartridge attached to a linear stage. Similar to how a human technician would perform dissection, the robot will drag the mosquito into a slot and place the mosquito’s neck into notches cut in two parallel dissection blades. An overhead camera will be used to provide computer vision feedback of this process. The blades will be actuated, cutting the head. After disposing of the mosquito’s head, the robot will return to the ring which will have rotated to present a new mosquito. The linear stage will index laterally immediately after the mosquito is cut. As additional mosquitoes bodies are

positioned on the cartridge, the linear stage will translate and expose mosquitoes to several stations at which the exudate can be squeezed out and salivary glands collected.

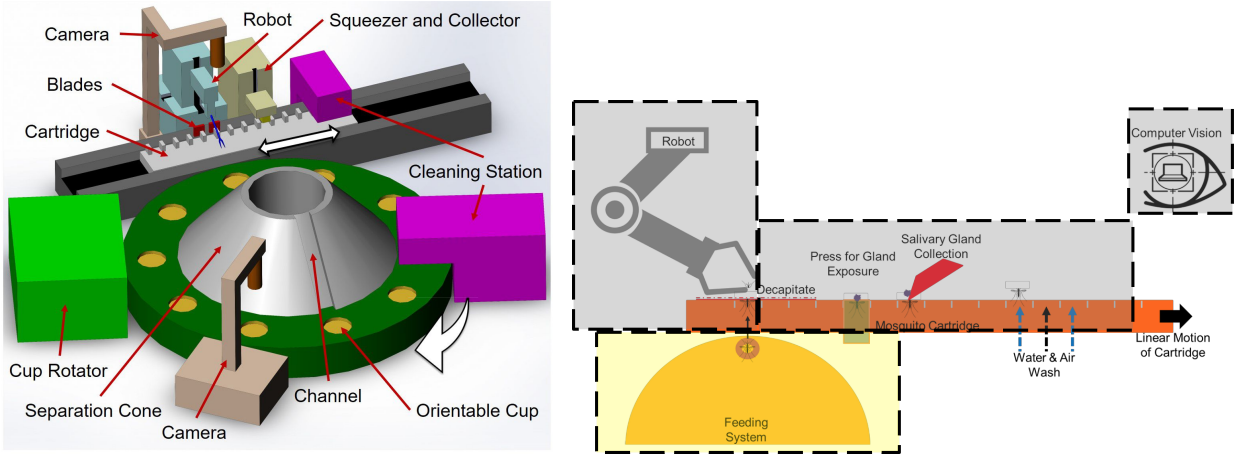


Figure 1. Two *concept image for automated mosquito dissection system. 3D concept (left) and 2D concept (right).*

C.Summary of Project Goals

The goal of the project is to engage in the design, development, and fabrication of several mechanical subsystems of a larger automated mosquito dissection system to aid in the production of malaria vaccines, as well as develop the software and methodologies needed to control and integrate these subsystems with one another and into the larger system. Specifically, we aim to:

1. Develop a robotic pick-and-place system for freshly-sacrificed mosquitoes
2. Develop an automated assembly-line-like dissection system to decapitate mosquitoes and extrude and collect salivary glands
3. Demonstrate the mechanical integration of these subsystems with one another
4. Demonstrate software integration with several computer vision algorithms developed by our collaborators to locate mosquitoes
5. Design improvements to these subsystems, particularly the dissection system to improve efficiency

Specific project deliverables are detailed and evaluated in Section V of this document.

II. Hardware Development

A. Robot and Gripper

In this project, we use a general-purpose robotic system to conduct our experiments. The system is displayed in Figure 2. In the future, a task-specific robot will either be purchased or custom-designed for the task, but while the design is still changing, using a high-precision robot that was already in the lab was practicable. This robot is a 4-DOF, linear stage robot by New England Affiliated Technologies, Lawrence, MA. A dual-axis X-Y stage is the robot's base and a Z axis is mounted orthogonally (NEAT: XYR-6060 and NEAT: LM-400, respectively). The robot also has a rotary axis which we do not use. Each axis is driven by a 12V DC servo motor over a 100 mm length lead screw. Axis travel is encoded with an incremental encoder. Overall positioning resolution of these axes was measured with a dial indicator to be approximately 10 micrometers. The entire assembly is mounted to an optical table. Robot motion is driven by a Galil controller (DMC- 4143), interfaced to a Linux computer by ethernet connection. For our project, we wrote a custom python package to communicate and interface with the robot. This package is called RobotMove and is detailed in a following section.

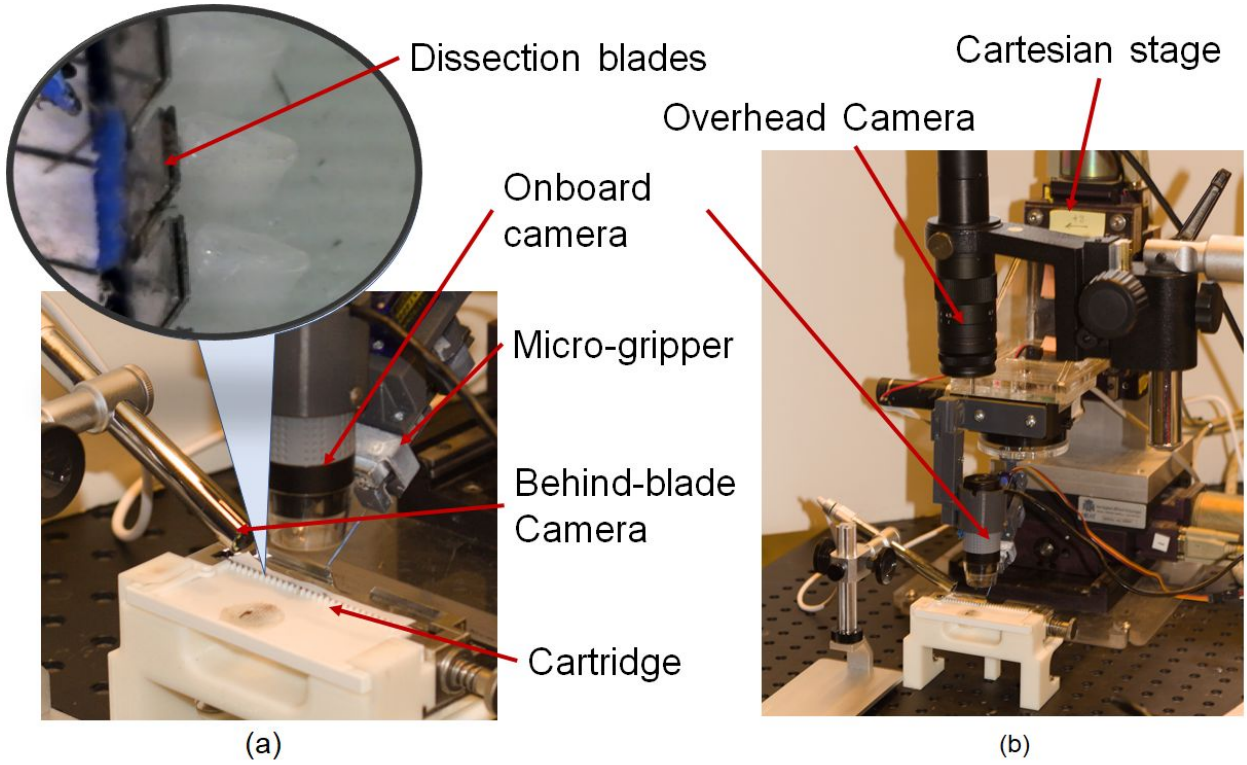


Figure 2. *Robotic system used in this project. Included in this figure is the manual dissection apparatus used in early tests. This figure is from [16].*

We have developed a gripper that is attached to the robot (Figure 3). The gripping mechanism has been salvaged from a Alcon Grieshaber tool, designed for tasks such as membrane peeling in retinal surgery. The surgical tool parts we use consist of a metal micro-gripper rod that is normally-open, as well as a metal sleeve that, when slid forward over the micro-gripper component, will close it. These pieces are separately attached to several custom-designed 3D printed pieces that are attached to a linear guide. A cam mechanism, powered by a model-airplane-style servo motor (HexTronik HXT900), slides the piece attached to the sleeve on the low-friction linear guide. This keeps the gripper itself in place while opening or closing, allowing the robot to need only to navigate to the position of the object it wants to grasp. The gripper is commanded via commands sent to an Arduino microcontroller over serial with the Linux computer. The Arduino has been programmed with the appropriate servo angles to result in cam movement to result in an open or closed gripper. This shared logic control is detailed further in the section describing the RobotMove package.

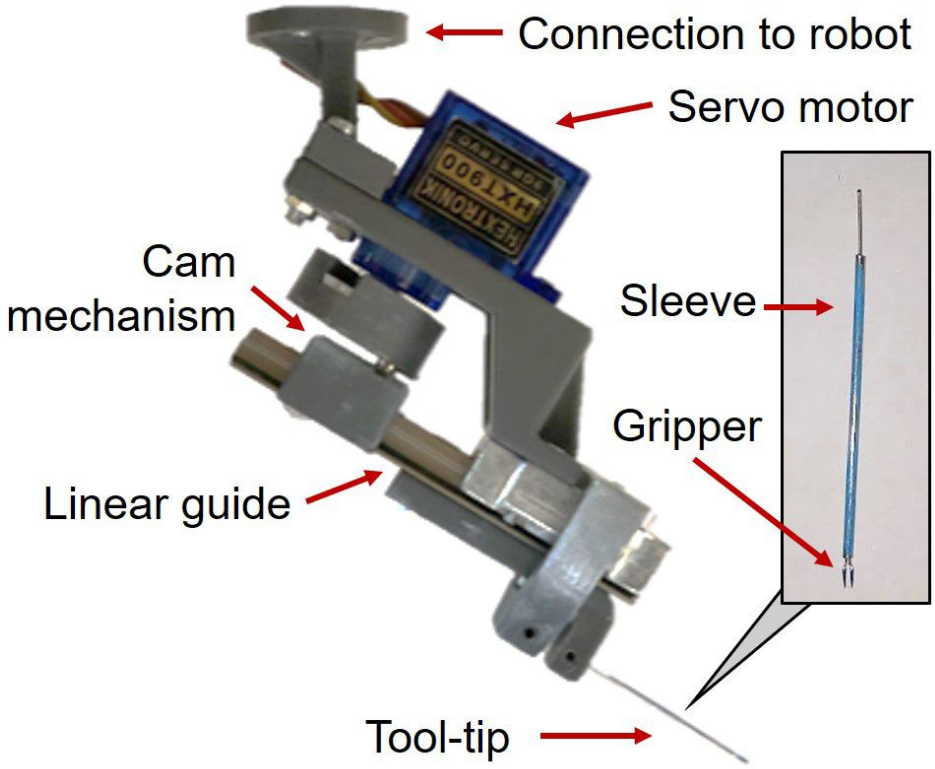


Figure 3. *Custom-designed micro-gripper used to grasp mosquitoes. Inset is a close-up view of the micro-gripper and sleeve parts taken from the Alcon tool. This figure is from [16].*

As a side activity, not formally in our project scope, we chose to further develop the gripper. While the gripper worked, the design is not optimal. It is quite bulky, and would require the final choice of a robot to be larger than may be needed. This is because the current gripper was originally a short-term solution to the previous gripper design that was plagued by high friction, expensive and easily damaged ultrasonic motors, and difficult assembly. We wanted to provide

design concepts for future iterations of this gripper, and also provide possible mitigation against our dependency on the continued practicability of using the current gripper. The goal was the take the general idea of the current system and simplify it. Currently, the system uses a simple rotary servo motor and a cam to actuate along a linear slide. The current plan is to use a linear actuator with PWM control similar to the servo motor and replace the servo and cam with this. The precision linear slide will remain in use to help constrain the system. There is significantly fewer components in this design, decreasing complexity. There are two blocks, one for holding the outer sleeve of the gripper, and the other to hold the inner tube with the tweezer end. The gripper block is mounted on the linear slide and attached to the linear actuator. This block moves along with the actuator, opening or closing the tweezer end of the gripper. The design is not yet finalized but is near completion, and is displayed in Figure 4. It will be designed to be machinable so that it can be manufactured about of stiff materials easily. It is also designed with the user in mind, making it easy to replace broken gripper components and install them more easily and with greater accuracy. Effort was made to ensure the tweezer end of the gripper will have a lot of clearance, allowing less restricted movement when moving in the space of the dissection system.

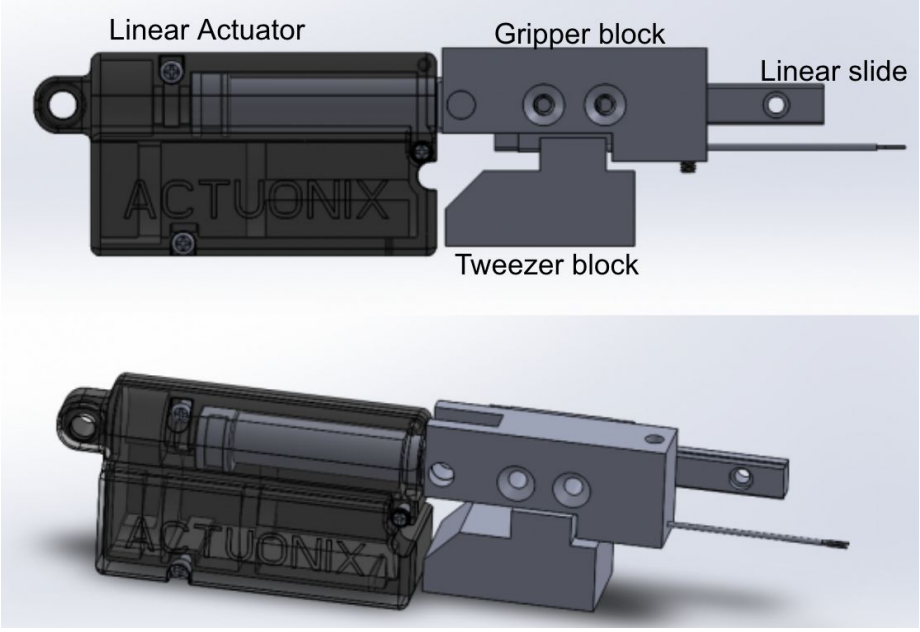


Figure 4. *Design for next-generation gripper*

B. Linear Dissection System

Manual System:

For early experiments, while an automated dissection system was being designed, manufactured, and tested separately, we used an existing, so-called semi-automated mosquito micro-dissection system (sAMMS) device for testing with the robotic system. This device, initially developed by a team at Hopkins and improved for production by Keytech, is currently

undergoing GMP (good manufacturing practices) validation to be rolled into Sanaria's manufacturing processes. This device is pictured in Figure 2.

Automated System:
The automated dissection system was developed in two forms. A longer version was developed that is intended for full functionality and eventual permanent integration, as well as a short version that was developed for the purpose of understanding the functionality of complex and error prone components. Both of these systems are depicted in Fig 4 with the major difference outlined.

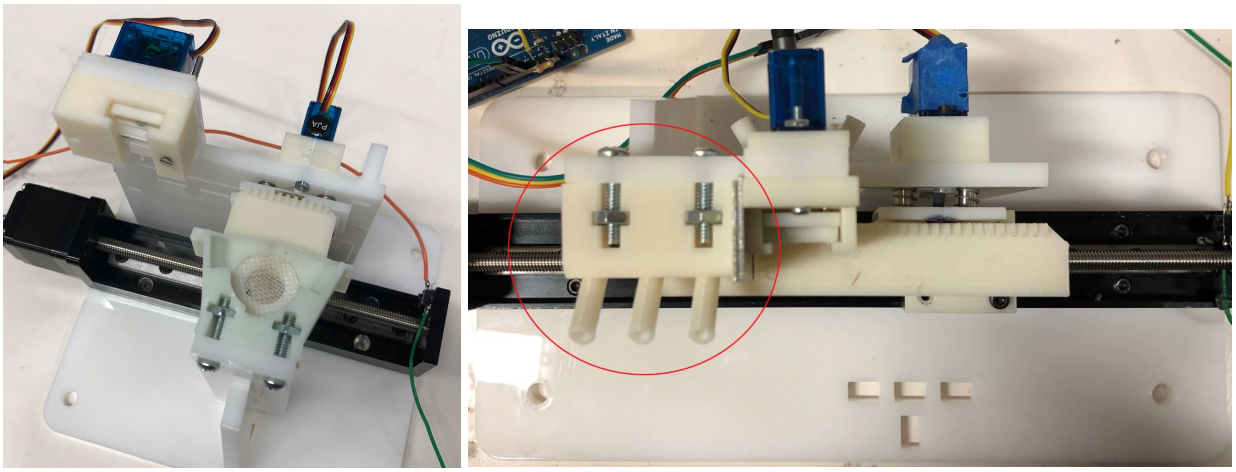


Figure 4. *Short, most current dissection apparatus version (left) and longer version of the dissection apparatus (right). Difference between the two assemblies is that the longer version contains that body disposal component which is outlined in red. Both accommodate a mock staging component but this is not shown on the longer version.*

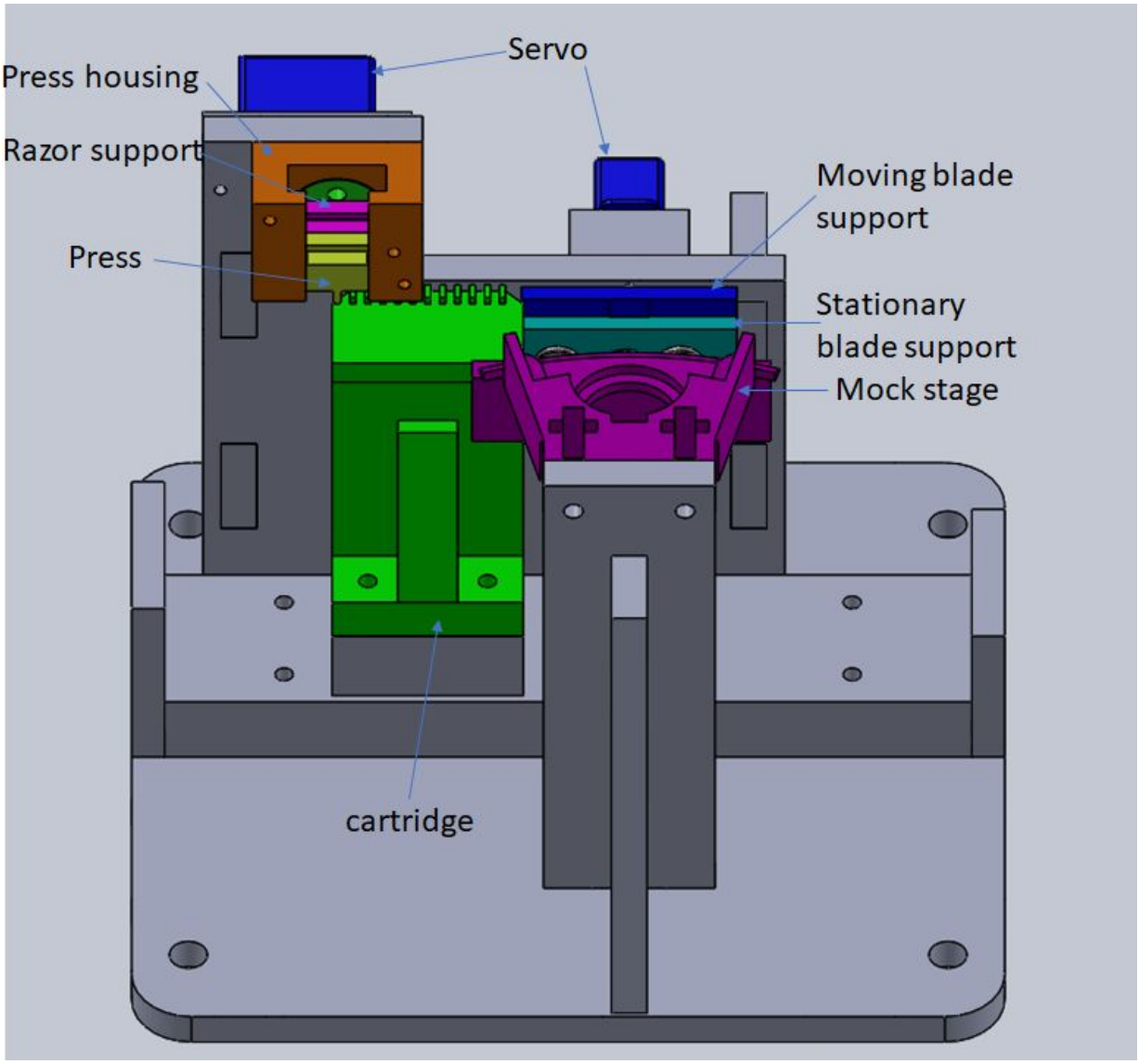


Figure 5. *Dissection system as a whole and the associated key components labelled.*

The **Dissection System** consists of a moving stage rigidly attached to a cartridge. Dissection is an intermediate step of the mosquito microdissection process. This step assumes that the mosquito has be successfully placed between the blades of the decapitation mechanism by the robot, in such a way that the body is aligned with slots, constraining the mosquito for subsequent steps. The dissection process consists of 6 discrete steps: robotic placement of mosquito (dependent on the robot), decapitation of the mosquito head, extrusion of exudate by squeezing, dissection of the exudate from the mosquito body by razor (shaving off exudate), exudate collection by vacuum, and finally body disposal such that the cartridge can be reused for subsequent mosquito processing. It is important to note that two systems were built such that simultaneous development and testing could occur. One stage has a travel length of 200 mm while the other has 100 mm. The most current design of the apparatus at the time of writing this report is on the short stage (depicted in Fig. 5), but the longer stage is intended for

incorporation in the final design. Furthermore, the short stage does not accommodate the cleaning step as this step was considered rudimentary and one that will not require significant testing. The cleaning component is outlined in red in Fig. 4.

The **Stage-Cartridge Assembly** enables for the relative linear motion of correctly positioned mosquitoes from module to module in an assembly line fashion. This consists of a commercially available 200mm of travel linear stage (Toauto 200mm Length Travel Linear Stage Actuator) that is rigidly mounted on the same support structure as all of the modules. Atop the linear stage is a cartridge with notches that are 1.5mm tall patterned 2.5 mm apart. The walls are 1mm wide and are placed such that the empty space between two subsequent notches is 1.5mm. This 1.5 mm pocket is where the mosquito is intended to rest during the dissection process.

Along the linear stage are a series of **Modules** which are the active components that execute the microdissection. All of the modules are contained on a single backplate that is rigidly attached to the same base that the linear stage is mounted upon. All of the module components are driven by servo motors (HS-55 Feather Servo Motors). The details of each module will be described in detail below:

The **Head Decapitation** module (a.k.a. cutting module) utilizes two stacked parallel blades that slide relative to one another laterally to cut the head off the mosquito. This module consists of two shoulder bolt, two springs, two 0.002-inch thick blades that are 6.75 mm wide and 12 mm tall, two 3mm thick blade substrates (one stationary and one moving), and a servo motor actuated cam that will laterally actuate the moving substrate. Furthermore, two ½-inch length shoulder screws are mounted on the module backplate with the substrates and springs being held up by the screws.This assembly suspends the substrates near the head of the shoulder bolt to provide an operable distance between the backplate and where the blades are to be positioned. The substrates are etched during laser cut manufacturing so as to provide a position for the blades to be glued into. The cam mechanism utilized in this module generates the lateral motion of the moving substrate relative to the stationary one. As such, a 3 mm diameter cam is rigidly attached to the servo motor, with a moment arm of 5 mm, and positioned into the moving substrate slot. When the servo motor is driven, the substrate and blade move and produce the desired cutting motion. The assembly can be visualized in Fig. 6 below and it is recommended that prior to attempted reproduction, the assembly in solid model and images be studied in detail. This module has demonstrated rigorous success and reliability through testing.

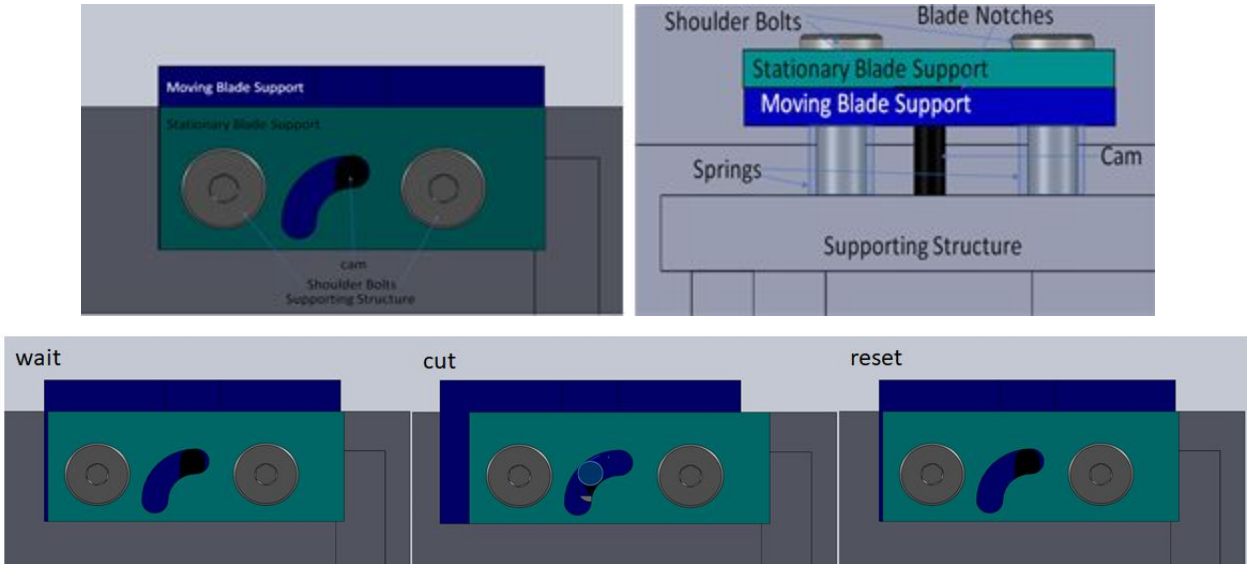


Figure 6. *Labelled front (top left) and top (top right) schematics of the dissection apparatus. Workflow and operation visualization of the cutting component on the bottom showing the substrates in their passive positions on the left and right and the actuation motion of the moving blade in the middle.*

A single module is responsible for **Exudate Extrusion, Exudate Microdissection, and Exudate Collection**. This module is mounted on the same backplate as the decapitation module and requires adequate alignment of the backplate with the cartridge. This module consists of a press component, razor blade substrate, razor blade, housing, cylindrical cam (with two 3 mm dowels positioned 45 degrees apart), vacuum needle, and servo motor. This component achieves its function by having two components interacting with same rotating cam component by means of slots. As such, the cam is mounted onto the servo motor and the dowels are positioned at different radii from the center of the cam and at different lengths protruding away from the motor. The housing provides normal force to the substrate and press to constantly maintain alignment of the components in the lateral plane enabling motion only in the vertical plane. Additionally, the slots on the press and substrate make it possible for the lateral motion to be decoupled but maintain the vertical motion of the components when actuated by the cam. As such, by interfacing the substrate on the external and shorter dowel, and the press on the internal and longer dowel, different vertical motion is observed between the two effector components. The assembly is initialized such that the press will move a greater distance first, followed by the substrate that lags behind the motion of the press. This construction of this component and the anticipated motion is depicted in Fig. 7. This accomplishes two steps with a single servo motion motion, first pressing/squeezing the mosquito and then almost instantaneously the microdissection of the exudate by the razor blade from the mosquito body. The collection aspect has been accommodated for a by a precisely

placed 16 gage needle attached to vacuum that will collect the exudate from the razor blade. We have experienced immense difficulty in the development of this module. While the mechanical components have demonstrated rigor and success, poor understanding of the mosquito and its relevant properties have brought up immense challenges in the fine tuning of this module. The poorly understood mosquito properties including stickiness of exudate, never before attempted methods for dissection, and reduced access to fresh (<4 hrs after sacrifice) mosquitoes have made the application of this robust mechanical system difficult for the task at hand.

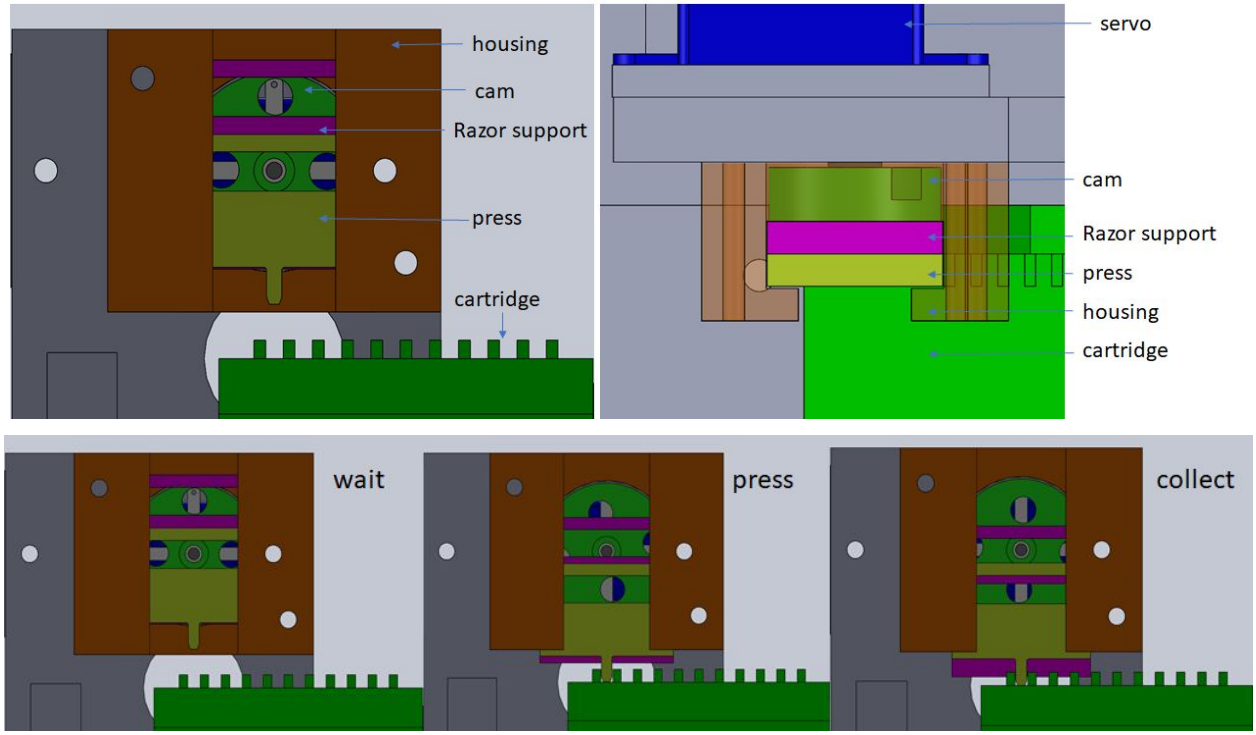


Figure 7. *Key components of the squeeze/collect apparatus in front view (top left) and top view (top right). The three motion components are shown on the bottom with the assembly passive on the left, the pressing motion depicted in the middle, and the collection microdissection step shown on the right.*

The **Body Disposal** module is responsible for removing the mosquito bodies after the dissection, exudate extrusion, and collection step. This system was designed and hardly tested because of the simplicity of design, ease in changing design parameters, and lack of time for its implementation. This design utilized three 16-gage needles that are pointed at the midpoint of the cartridge slots (where the mosquito should be positioned). The needles use positive air pressure in the two outside needles and a stream of water in the middle needle to first remove the mosquito body with air pressure, then wash the cartridge with water, and remove excess water with air pressure provided by the third needle. This step enables the automatic reuse of the cartridge upon the completion of this step.

C. Design Concept of Rotary Stage

The overall goal of our project is to automate the dissection of glands from mosquitoes to streamline the processing of a malaria vaccine for Sanaria. The reason we decided to design a rotary stage when we already have a linear stage is to further optimize our solution. Currently we have a linear stage with three subsystems (cutting, squeezing/gland collection, and washing).

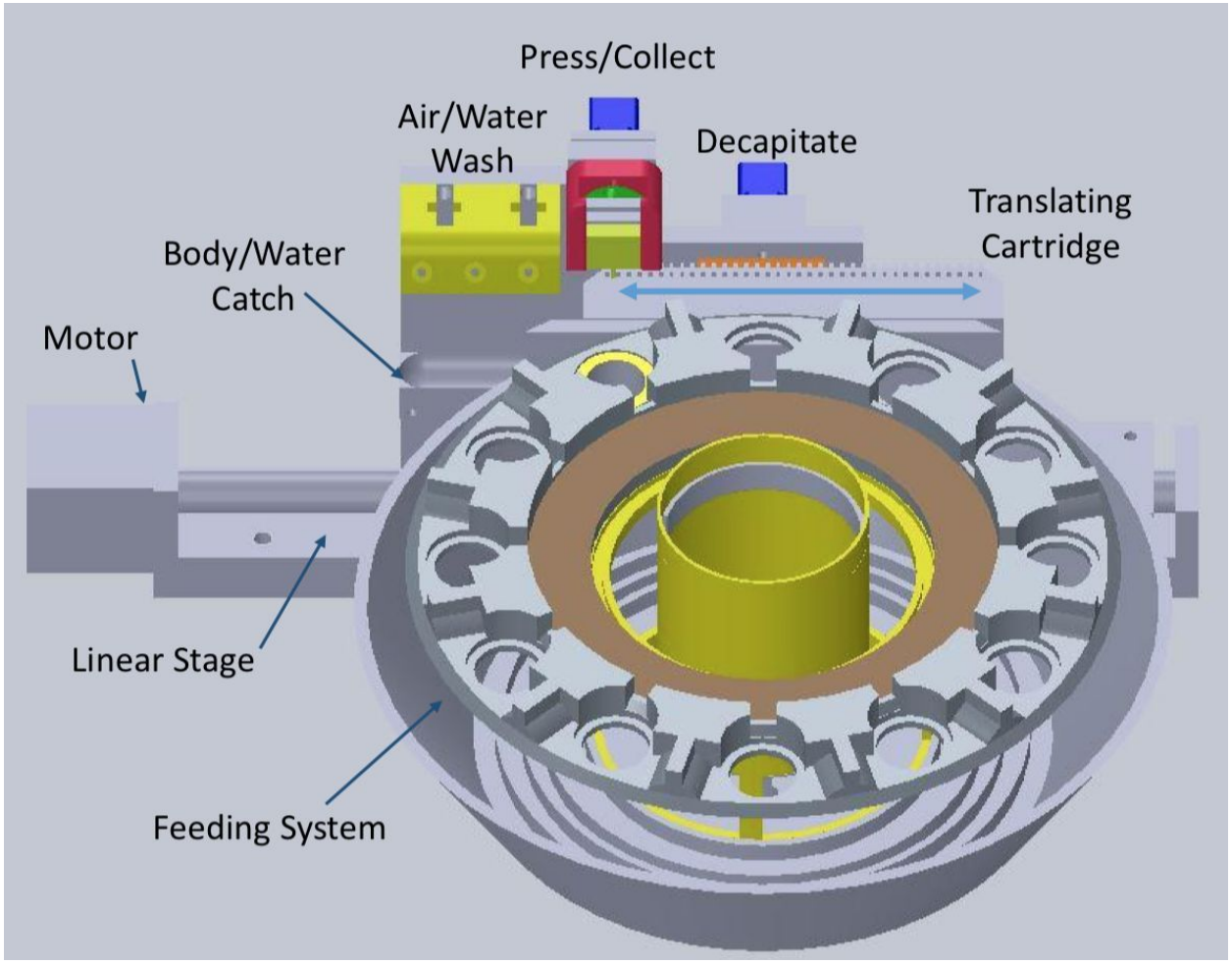


Figure 8. *The Linear stage design currently being used to process mosquitoes with its subsystems and the feeding system used to present mosquitoes to the robot to place on the cartridge.*

The issue with this system is that it is not continuous, once the cartridge holding the mosquitoes reaches the end of the linear stage it has to return to its home position wasting valuable time not processing mosquitoes. The obvious solution is to take the existing subsystems which are developed on the linear stage model and transplant them seamlessly onto a continuous rotary design. This allows for continued testing of our subsystems and the downstream dissection

process without major design changes to the existing setup. Which allows for tuning of the subsystems without needing to integrate the rotary stage which will likely take weeks to construct and test.

There were two obvious design avenues for the rotary stage approach; the **Concentric Rotary Stage**, and the **Tangential Rotary Stage** (see Figure 9). The concentric design, which is designed to exist as the concrentic outer ring of the system with mosquito feeder at the interior. This design was terminated early due to the scope of the project, with the issue that the concentric system was so dependant on other teams design choices. Given the time to do this project and the number of different components being designed for the system it did not seem reasonable to try to design around a constantly changing feeding system and its associated subsystems. However in the future the concentric design does offer benefits over the tangential design, in that it can be more compact and could possibly have two processing systems for a single feeder (Figure 10). As the larger project comes to a close, it may be beneficial to re-investigate this method.

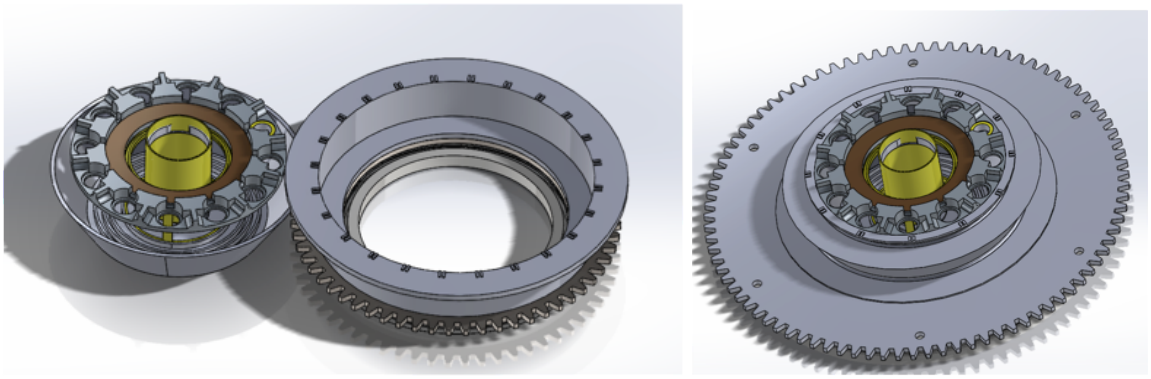


Figure 9. *Initial concept designs for tangential (left) and concentric (right) rotary stages. These where conceptual designs that lead to a decision of pursuing the tangential design in favor of concentric.*

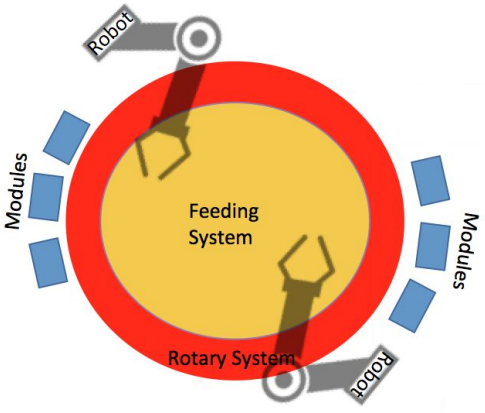


Figure 10. *Rough idea of how the concrete design could utilize multiple robots for a single feeder and process multiple mosquitoes simultaneously.*

The Tangential Rotary Stage design is what we currently have a more complete design review of. The current design utilizes the same subsystem currently being developed with the linear stage design (see Figure 11). This again is to streamline the process of integrating the new new design in the future when we have thoroughly developed method of extracting the processing the mosquitoes.

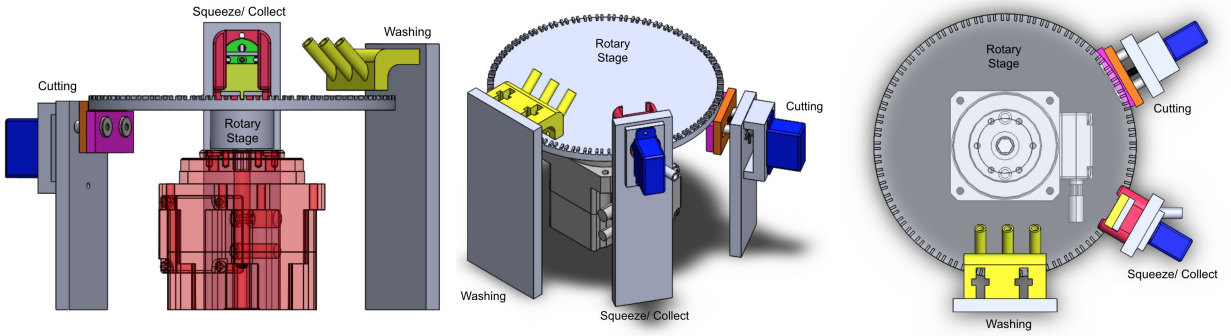


Figure 11. *Rough design of the three subsystem around the tangential rotary stage. Front view (left), Isometric view (middle), and top view (right). Rotation of stage is clockwise as viewed from the top view.*

A significant reason the design was chosen was due to how it can be integrated with the existing feeder system. Since the design is tangential it interacts in the same way the linear stage currently does with the feeder system. The linear system is something the other designers are familiar with and have already designed components with it in mind. The design is simple and has a relatively small footprint, it easily integrates the existing subsystems since the stage is basically the same as the linear cartridge but wrapped into a circle (Figure 12). Since we want a high degree of precision in this system a harmonic drive is being used. This may not strictly be necessary but it is believed to only have added benefits. The diving motor is a simple stepper motor with an absolute encoder that should be easy to integrate with our existing setup since our linear design also uses a stepper motor, however a 24V power supply will be needed for this new motor. The harmonic gearbox will make our system a zero backlash system and having a homing mechanism design into the system will make an extremely robust and accurate system for locating the slots directly under the cutting and squeezing/gland collection subsystems. The stage is connected to the driver via a hex shaft which mates the two rigidly. For this design the current robot setup will not work. This design only works for an inverted robot design mounted above the work space. The robot will need to overhang the two systems and perform its pick and place operation from above. This requires a new robot to be designed but both the concentric and tangential systems have this same problem.

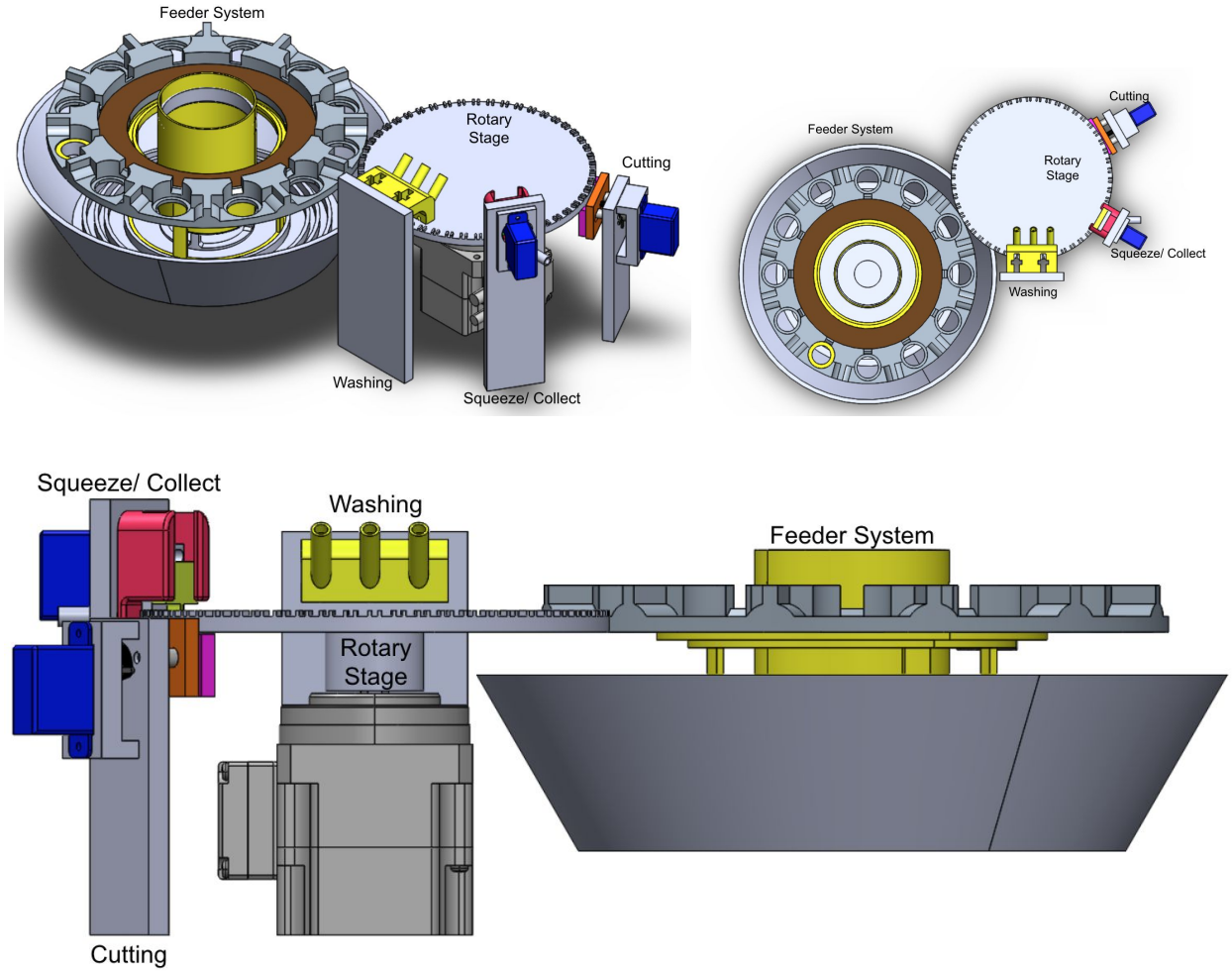


Figure 12. *Possible mating of tangential rotary stage with existing feeder stage. This is just a suggested setup with the idea that the overhanging robot would not need to rotate after picking up the mosquito to place it but other configurations are possible where the cups are closer to the cutting mechanism. Isometric view (top right), top view (top left) and side view (bottom). Looking at the side view the feeder will need to be on an elevated stand but this is not seen as a significant problem which is simple to solve.*

This design was presented to one of our mentors for review. After meeting with them a small change was made to the design to increase stability of the system in key ways. The stage in the system above was not sufficiently supported for a proper rotary stage design so a simple support system with bearing surfaces was designed. There is a three legged table like design with a standard roller-ball bearing for axial alignment and a thrust bearing for smoothly supporting the rotary stage on the table, also it should be noted that the feeder was moved closer to the cutter to minimize the travel distance of the mosquito (see Figure 13, 14).

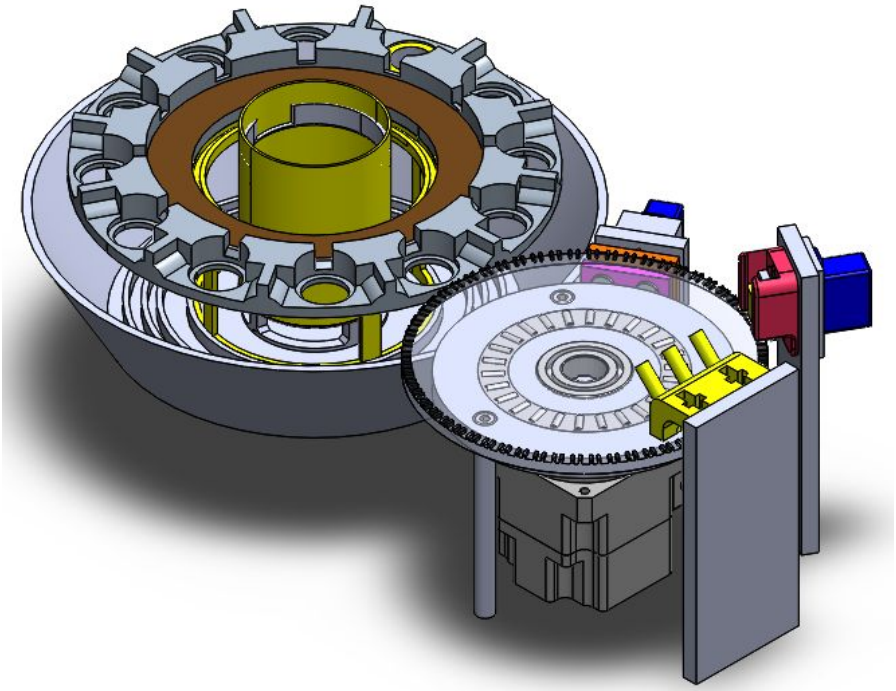


Figure 13. *Isometric view of the rotary stage design with support structure.*

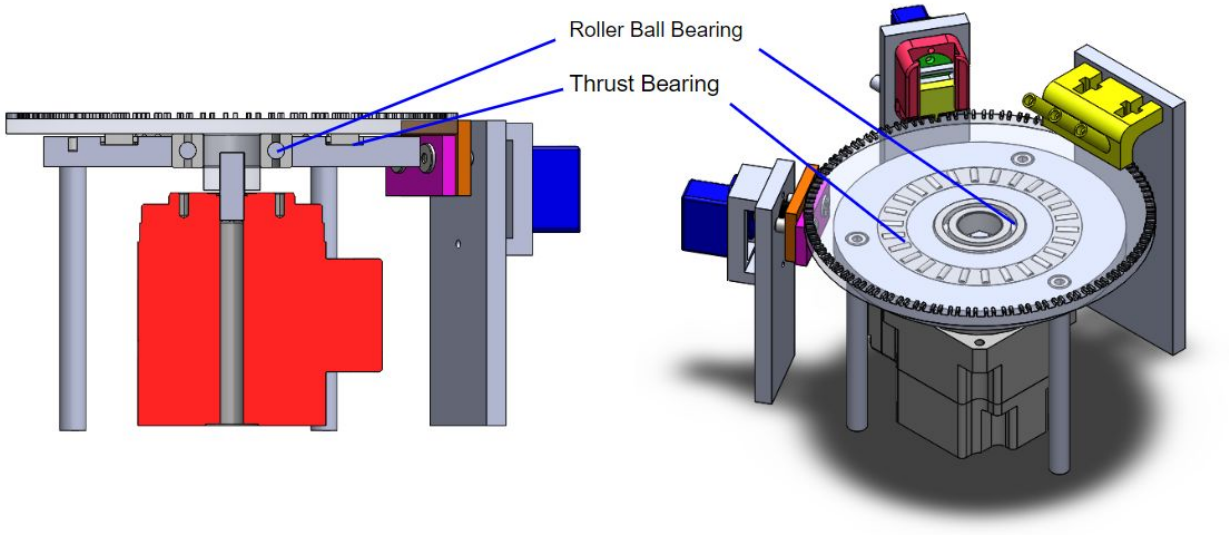


Figure 14. *Front section (left) and isometric view (right) of design additions to the rotary stage.*

This new design was then again presented to one of our mentors for review. During the meeting, a key problem was seen with the design that required a overhaul of the design in its entirety. A key design constraint was ignored in the previous design, it was neglected that the dragging of the mosquito is key in aligning it for dissection. With the prior design the way the

mosquito would need to be dragged is crude and not suitable for our system.The main issue with the last design was that the mosquito needed to be dragged either quite a long distance and rotated which is not practical nor efficient. To remedy this a new design was suggested where the stage was inverted such that the subsystems are interior of the circular stage with and octagonal cut out (see Figure 15). This edit keeps some of the design lessons from the last design in mind. We have replaced the custom rotary stage and the harmonic drive with a precision rotary stage which can be commercially sourced and added our mosquito stage directly on top of it. This replacement of the drive mechanism will remove the need to design our rotary stage constraints. The stage consists of two plates. The first is a base plate that mates to the rotary stage directly (shown in white) it is designed to mount the stand offs and pusher cam guide (show in pink), which will be discussed later. The second plate (shown in green) sits on the stand offs and is where the mosquitoes will be dragged into the slots to be processed. This plate is designed with 8 cartridge slots for placing the mosquitoes to accommodate the cutting mechanism with minal design changes from the current linear stage design each slot exists on a flat of the octagonal cut out. Each of the subsystems is designed to be overhanging as before but now are placed in the interior of the cut out as mentioned earlier.The design of the cleaning unit and the squeezing & collecting unit have not changed other then the manner in which they will be mounted. The cutter unit is the only subsystem that posed a new design challenge.

The cutter unit is now mounted via spring loaded guides so when the system rotates the cutter will move out of the way of the stage and not stall the system. Originally an additional actuator was going to be used to move the cutting mechanism but some members were wary of introducing another action unit to the design. To remedy this a cam design was implemented (shown in pink), though currently this is only a rough design concept in need of several changes it is believed to be a design headed in the correct direction. The cutter carriage has a peg that interfaces with a circular cam (seen in pink) which is designed to push the cutter away from the cartridge plate as the system rotates. This design choice may also prove to be better than the original idea since now the timing of the movement is dictated by a mechanical feature instead an actuator controlled in software.

We plan to interface with the feeder system in a similar way and the original design (see Figure 16). The robots orientation will be the same as suggested for the prior design as well (see Figure 17). The current setup will need to reoriented 90 degrees. Currently we have the gripper move forward (away from the robot base) and backward (toward the robot base) we need it to be oriented so that the gipper movement is left to right of the base.

This design is far from a finalized version and requires some modifications before a physical prototype should be made, but overall it serves as a strong starting point. It includes most if not all the needed components and how they interact with the stage and feeder system. What is not included is a detailed method of fixturing each of the overhanging subsystems but this is something that is relatively easy to design as a more concrete design is developed. The cam mechanism is crude at best and is missing crucial elements it simply serves as a general concept of what type of mechanism we are envisioning. The design though incomplete is close

to something that we think can be quickly prototyped and iterated for future development of this system.

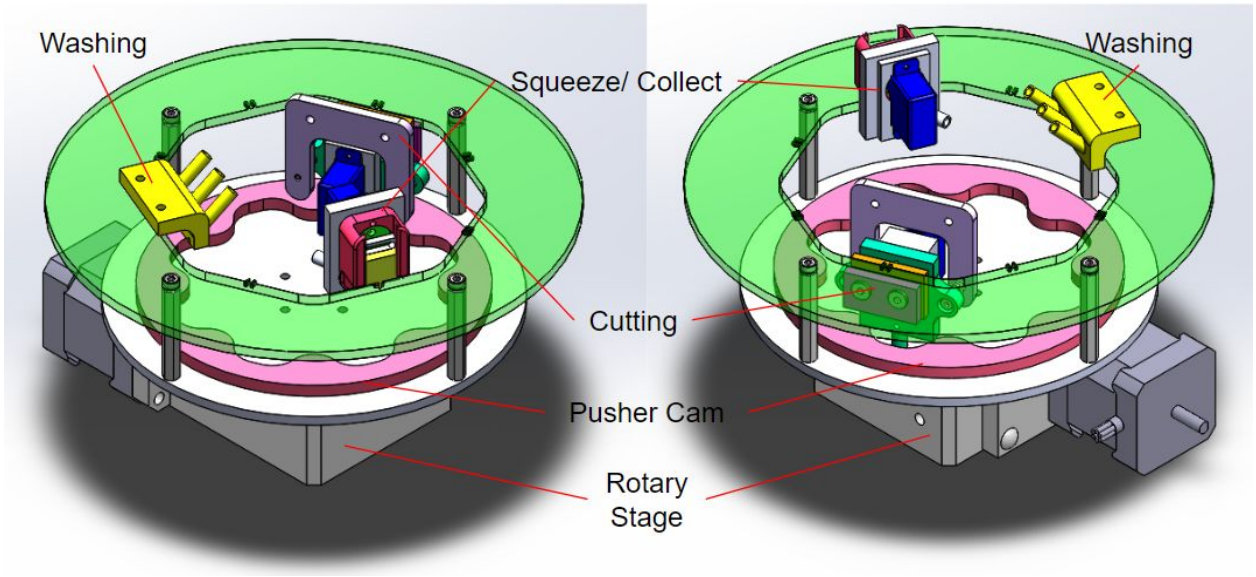


Figure 15. *Rotary table design revision to accommodate mosquito dragging, two isometric views of the inverted rotary stage design highlighting different key features.*

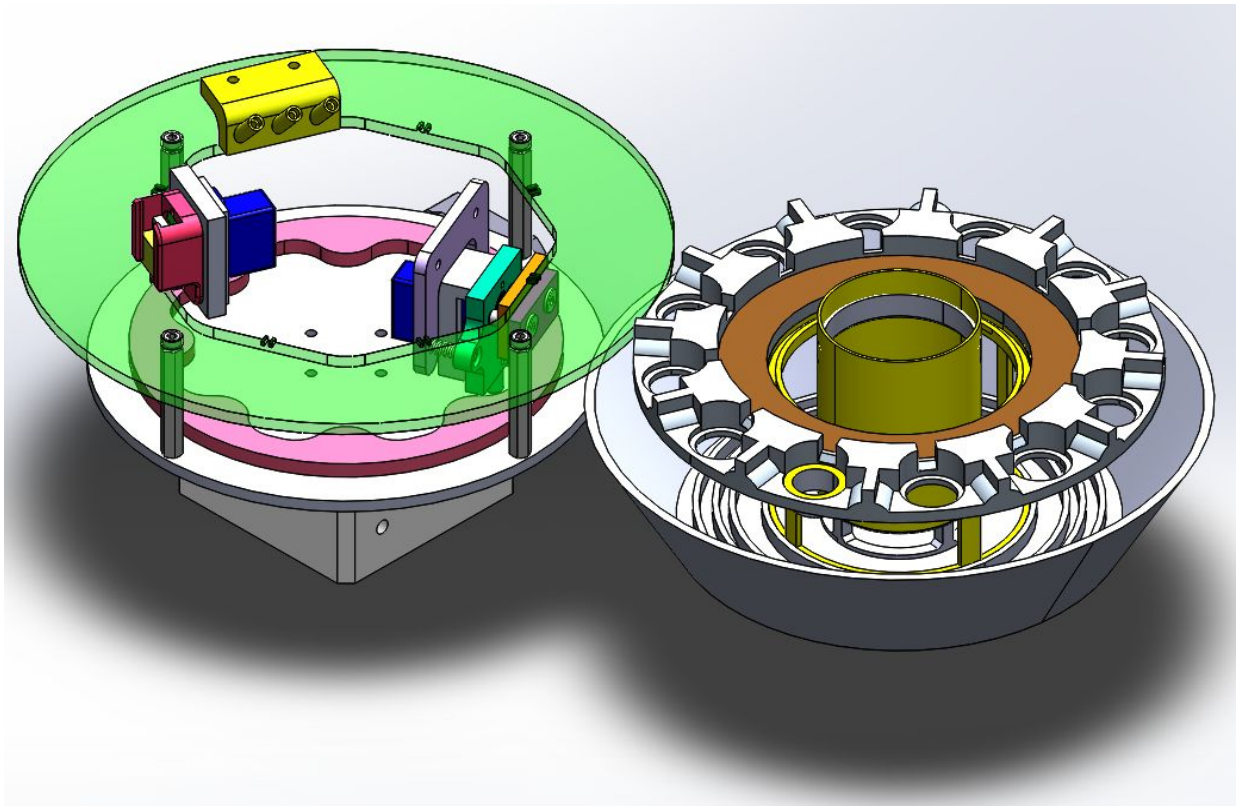


Figure 16. *Isometric view of the new system and how we plan to interface with the feeder system.*

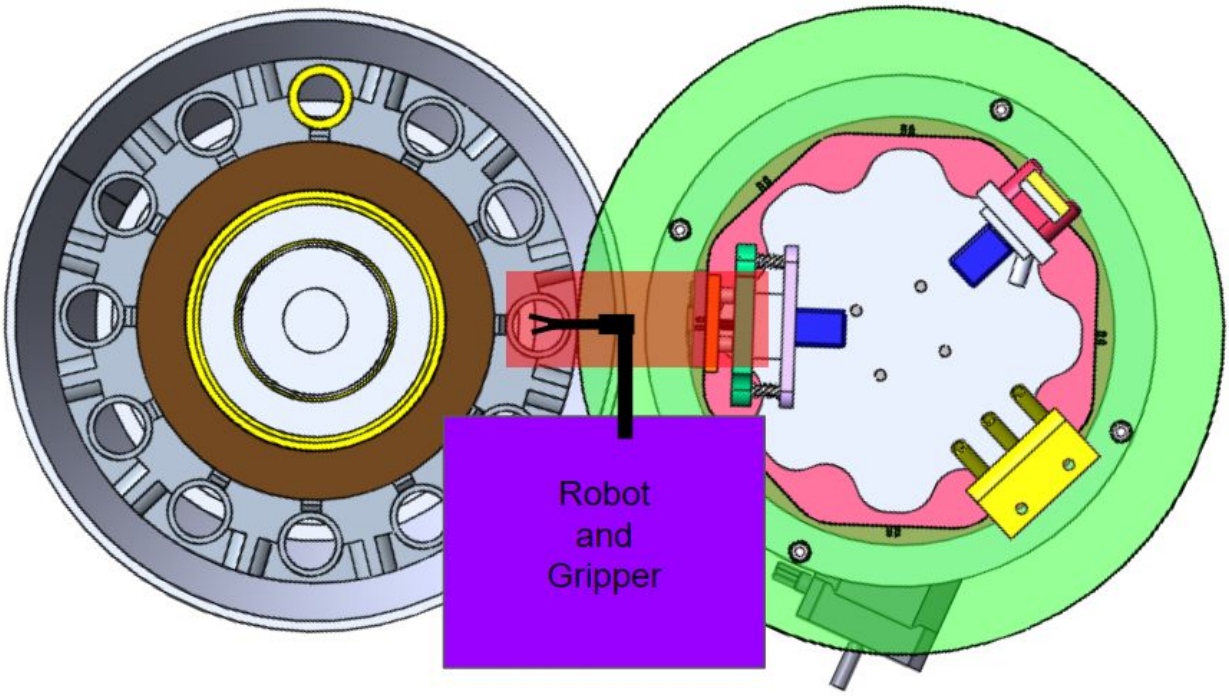


Figure 17. *Rough idea of robot position relative to the system. Highlighted in red is the relative workspace of robot gripper end effector.*

III. Software Development

A. Architecture

This section details the design of the software used to calibrate, control, and automate the robotic pick-and-place and dissection systems. A block diagram of the software structure is given in Figure 18. At a high-level, the systems are controlled using Python scripts running in a Linux machine. There are distinct scripts to perform distinct tasks, for example, a separate script is used to calibrate the system than to manipulate mosquitoes. However, each of these scripts make use of utility packages to perform tasks such as computer vision or communication with hardware. All actuator commands go through a low level controller - the robot is controlled through a Galil controller, while the servos and steppers involved in the robot gripper and dissections systems are controlled via commands from an Arduino. We introduce the main workflow of the automated system and detail several important components in its realization, namely calibration techniques, the utility packages that allow for software - hardware interaction, and the integration of our system with computer vision algorithms. These computer vision algorithms developed by our collaborators Prasad Vagdargi and Hongtao Wu that are not members of this project. While we did not develop this code, we were responsible for using it so that it could be used to automate our physical systems.

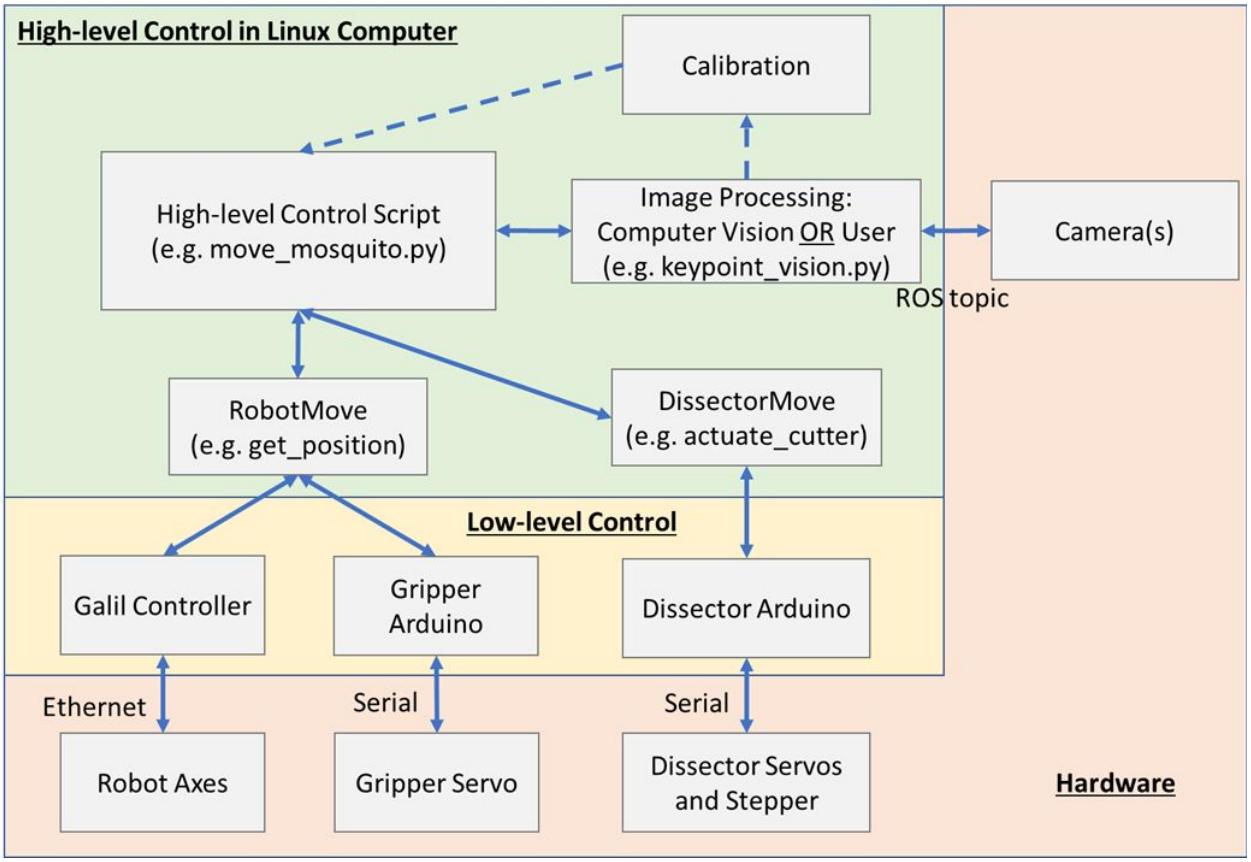


Figure 18. *Block diagram of software architecture and the interfaces with hardware*

B. Calibration Procedure

A “hand-eye” calibration must be performed to find the transformation between robot and camera coordinates in order to determine where to move to grasp a mosquito. We implement the common calibration technique of commanding the robot to travel across a grid within its workspace, recording its position in both coordinate systems, and then using a Bernstein polynomial fitting to fit a mapping between these two systems. In our case, this has the added benefit of correcting for any systematic errors in robot movement or camera lens distortion.

Earlier students on the project had developed a computer vision technique to find the robot’s tooltip, when painted blue, in an image. We had to improve upon this technique as it was not very robust. Any objects in the background such as the robot when it moved into view, or even the shadow of the gripper could produce false positives. We discovered that we could greatly increase the chance of accurate detection by cropping the image into a small region where the tooltip would be mainly featured. This was achieved using the procedure displayed in Figure 19. A region (denoted by the red border) is defined in which the robot will move in a grid. A grid size is selected (e.g. 100 points in a 10x10 grid). The tooltip position is determined at the starting

location, and after one lateral and vertical movement. From there the displayed grid is generated which estimates the approximate point at which the tooltip will be at each point as it moves through the commanded grid. The full image is cropped to just a small area around this point (right side of Figure 19). We observed accuracies less than 50 microns using this calibration technique. A detailed walkthrough for performing the calibration procedure is given in the next section.

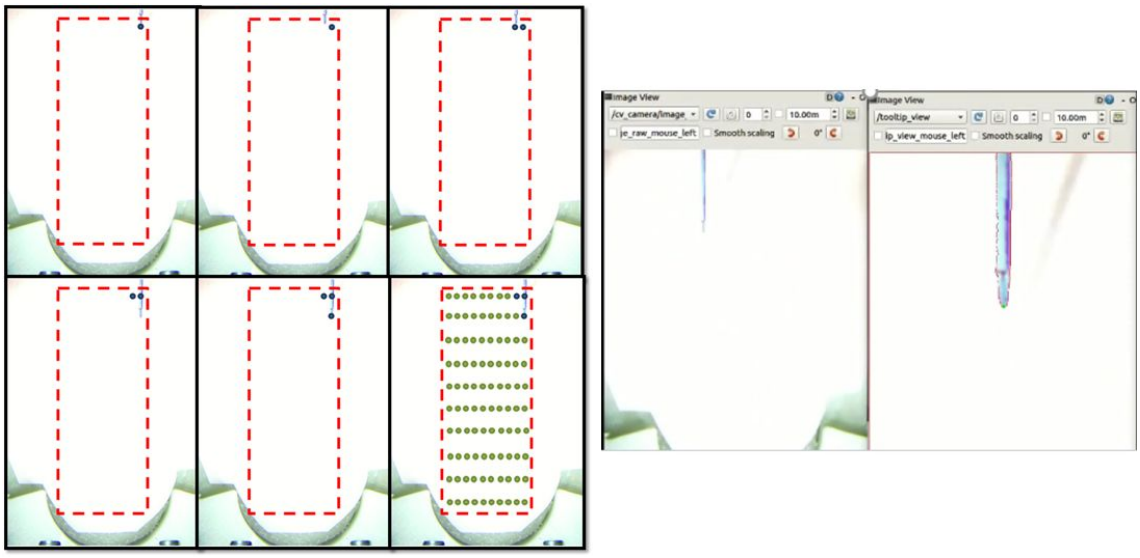


Figure 19. *At left, a stepwise demonstration of the estimation procedure of robot tooltip location. At right a demonstration of using that location to crop the image down to mainly feature the tooltip. This led to improved detection and accuracies in the 50 micron range.*

C. High-level Automation Procedures

Identification of Mosquitoes (Image Processing)

We have used three main image processing procedures to detect mosquitoes and identify the location of certain features on their bodies. Two are based on the ‘deep network’ computer vision algorithms that our colleagues have developed. The first is a segmentation algorithm that can roughly identify a mosquito’s location from an overhead camera, but needs a high resolution, close-up image to perform the segmentation itself, determining the approximate regions in the image where the head, body, and proboscis of a mosquito are. The second method detects a bounding box around mosquitoes and identifies 7 key points on the mosquito’s body (Figure 20). The third procedure involves replacing computer vision with a human user, having them click locations on an image and having the robot and code respond as if that was the point determined by a computer vision algorithm. This user-driven image processing component was important both for early in the project, before the vision systems were developed, and later for unit testing just our mechanical systems.

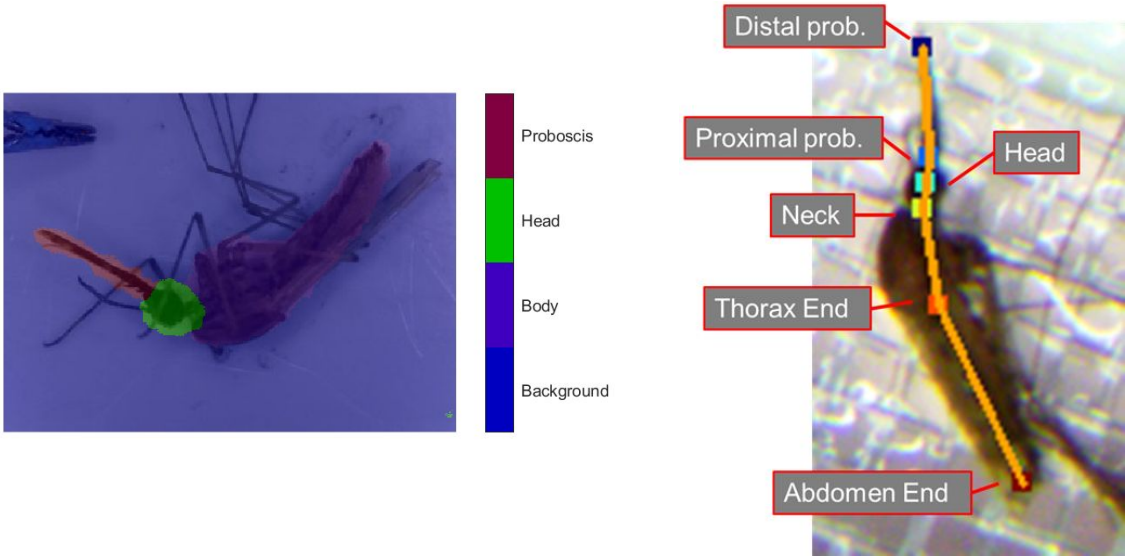


Figure 20. *At left the segmentation computer vision algorithm, at right is an example of the keypoint computer vision algorithm output with the keypoints labeled*

Robot and Dissector Motions:

As we integrated subsystems and vision algorithms the behavior of the robot had to be changed to accommodate physical limitations or requirements. Generally, we had three different implementations of automated motions that we developed, which will be presented in the order they were developed to demonstrate the progression of system integration.

The general procedure has remained fairly consistent throughout the project. An image is acquired of the workspace and a mosquito is located. The robot is commanded to move to a point above the mosquito's proboscis. Here, with an open gripper, it drops down to the level of the stage and closes the gripper to grasp the proboscis. The mosquito's head is slightly lifted above the stage and it is dragged towards a slot in the cartridge. The process of dragging the mosquito, as well as finally dragging it into the slot acts to straighten out its orientation. The neck of the mosquito is then identified in an image and transformed into robot coordinates. An offset between the neck and the location of the gripper tooltip is computed and the robot lifts the mosquito's head up, pulls it over the notch in the blades, and drops it down. If the alignment is correct, the neck will be within the blades such that they can be actuated and the head will be removed. At this point, the robot pulls away, drops the head, and is ready to collect another mosquito. When using the automated mosquito dissection system, the body can then be shuttled to a squeezing station where exudate, including the salivary gland, can be squeezed out. It should be noted that at this time, the height of surfaces, location of the slot, and location of minimum blade clearance must be determined in robot coordinates (encoder counts) via a manual calibration stage. In later iterations, it would be ideal to add some fiducials to the setup and calibrate to these, inferring the other values from the CAD models.

We began the semester without any computer vision algorithms, so we accomplished these tasks by having a user click on an image and reading in the pixel value they clicked on. We also used a manual dissection system, having not designed and manufactured the automated dissector at the time. The robot motion was simply what was described in the last paragraph. This motion had to be adapted when we integrated our system with the segmentation vision algorithm. As the camera needed to get close to the mosquito in order to produce accurate segmentations, we mounted a camera onto the robot itself in addition to using the overhead camera in the previous setup. We would detect the general region of the mosquito in the overhead camera and then navigate to the centroid of that region plus an offset so that the mosquito was in the image, but the gripper was not touching it. Then we could take an image, determine the proboscis location, grasp the mosquito, and continue with the next steps as normal. This robot trajectory is shown in Figure 21. This is the trajectory that was used to collect the data that we discuss in the manuscript we submitted to CASE 2019 [16]. To summarize these results, we tested 50 mosquitoes and saw 100% accuracy in grasping the mosquito and 90% accuracy in placing it in the correct place in the blades.

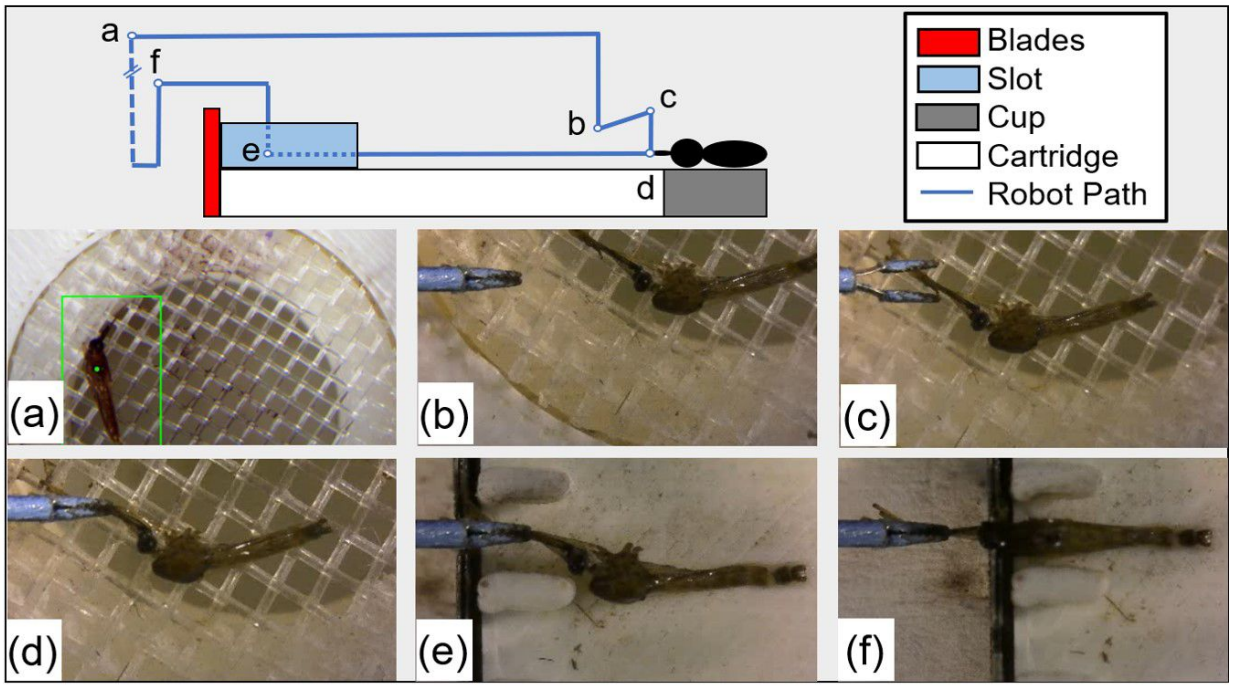


Figure 21. *Robot path is shown when using the segmentation computer vision algorithm and the manual dissection system (sAMMS). This method uses an on-board camera that has to get close to the mosquito to get a good image, hence the 'b' action in the figure. This figure is from [16].*

We then integrated the keypoint detection computer vision algorithm. This allowed some simplification in the robot trajectory, as all of the imaging could be performed with the overhead camera, cutting out the extra robot movement and image acquisition that was added for the segmentation algorithm. This was equivalent to cutting out waypoint 'b' in Figure 21. We only

did preliminary tests in this setup as the automated dissection system design was completed and we moved on to integrating that setup. Generally we saw great mechanical results and the vision worked well with picking up the mosquito, but not with determining the offset for correct placement.

Finally, we integrated the robotic system with the automated dissection system, having only used the manual system until this point. This system has a more complicated geometry that had to be accounted for, as demonstrated in Figure 22. With the automated system we were also able to automatically actuate the blades, which proved to result in cleaner, more consistent cuts.

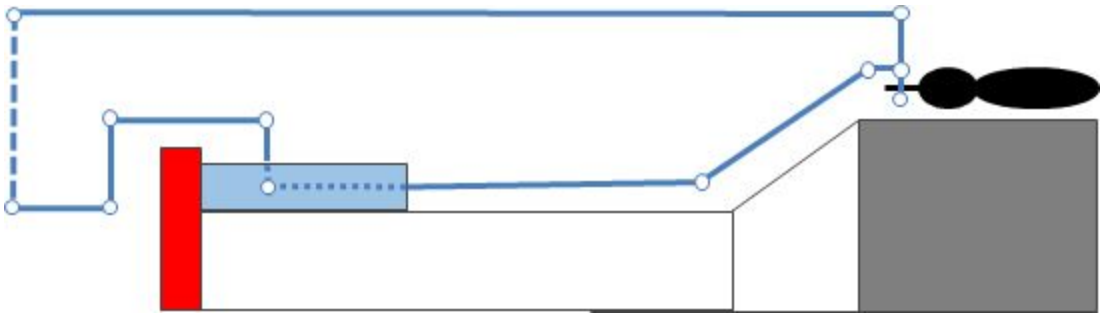


Figure 22. *Robot path with the integrated robot and automated dissector system using the keypoint vision algorithm. All of the image processing is done from an overhead camera, and thus a less complicated path is needed to pick up the mosquito, though the difference in stage-to-cartridge height has to be accommodated for.*

D. Utility Packages

The following are two utility packages that we wrote specifically for this project. A general overview of these packages are given here. Function-specific descriptions and input-output documentation is given in Appendix A.

RobotMove:
This custom-developed package is the conduit through which control of the robotic system and gripper is accomplished. The package is instantiated as a class and contains functions which allow for communication with lower level Galil and Arduino controllers to initiate movements. Additionally, several features for user convenience and system safety are implemented.

At first, this package mainly was implemented as a wrapper for the existing Python Galil communication library. Working with the Galil library directly is unintuitive, as commands are essentially in assembly language. At this early stage commands such as initializing setting and querying speeds, accelerations, gains, etc. were implemented. To physically move the robot, two commands were implemented for relative and absolute commands within the robot coordinate system where values are measured in encoder counts. As the project evolved, this package became the ideal home for several other feature improvements. The two most substantial improvements were the integration of the robot gripper communication protocol into

this package (and into Python and Linux entirely) and the development of a “Allowed Volumes” suite of functions that drastically improve the safety of the system when operated by new users, which will be crucial for the transition of this project to new students in the future.

Communication with the Arduino is done over serial communication. We essentially use the Arduino in this project as a cheap, easily integrable I/O board, with the added feature that it can share more of the programmed logic than a typical I/O board would. The high level Python code will only send a command to the Arduino that effectively represents a command to “connect”, “open”, “close”, or “disconnect”. When first connected, the Arduino is set into a waiting loop, with the servo motor deactivated without power. This prevents motor chattering that leads to early burnout of these inexpensive servo motors. Once the Arduino has been sent the “connect” command, it then waits for commands to either “open”, “close”, or “disconnect”. The function of these commands are straightforward. It should be noted that the angles to which the Arduino moves the servo to drive the cam to either an open or closed position are stored locally on the Arduino’s flash memory. This logic sharing decision involves a trade-off between user transparency and ease of tweaking these values, with system safety as commanding the wrong angle could damage the system.

The final notable feature in the RobotMove package is the inclusion of a user-friendly way to add, remove, and enforce “allowed volumes” in which the robot can be commanded. This feature is defaulted on, and will prevent robot motion if any requested move of the robot would land it outside of any predefined allowed volume. The user adds one of these volumes by selecting two corners of a rectangular prism in the robot’s coordinate system (Figure 23). These values would be given in encoder counts. Any rectangular prism in the space can be defined by a corner with the lowest dimension of the space for each axis and a corner with the highest. These values are saved in a file and is loaded into class variables when RobotMove is instantiated in a higher-level script. A simple check is done against these regions before any commanded movement is sent to the Galil controller.

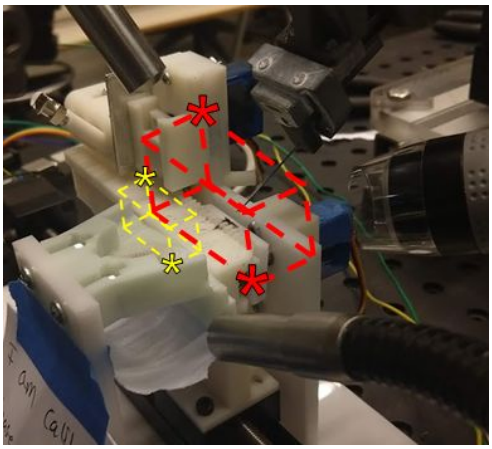


Figure 23. *Visualization of Allowed Volumes in the robot’s workspace. The asterisks represent the corners that the user would define these regions with.*

DissectorMove:
This package is based off of the RobotMove package, but provides serial communication commands to an Arduino to perform the low-level control on the actuators for the linear dissection system. The software design is shared between the two packages - it is instantiated as a class and contains functions that can be called in higher-level scripts to command motion. In this way, the dissection system and robotic pick and place system were integrated mechanically and could be controlled from a single Python script. Physical commands include actuating the cutter and squeezer mechanisms (which are controlled with servo motors) and incrementing the cartridge position (which is controlled with a stepper motor). In the future, the hardware control of the dissection system and the gripper would likely be combined onto a single Arduino or I/O board, at which time DissectorMove and RobotMove might make more sense as a single package.

E. Arduino Code

The Arduinos are essentially used as combination I/O ports for the PC and low-level controllers for the motors. Eventually, these functions would likely be moved into the robot controller, or some dedicated programmed logic controller (PLC) or data acquisition (DAQ) board, but the Arduinos are cheap and easy to implement at this stage of development.

They communicate with the computer over serial. There is a walkthrough in the comments at the top of the RobotMove package if there is issues getting this connection to work.

The microcontroller on the Arduino loops constantly. A connected_state boolean is off at startup. At each loop, the serial buffer is checked. If the computer has sent something, the Arudino reads this as a string. If the string containing the command to connect is received, the connected_state boolean is set and motors are powered on. Otherwise, the string is ignored. Once in the connected_state, anything sent is assumed to be a command. These commands each will run a corresponding set of code programmed in the Arduino. The simplest is a command to disconnect, which powers off the motors, and puts connected_state back to False, returning the Arduino to the outer loop of rejecting all strings that are not the connect command. Other commands will instruct actuators to move, e.g. setting a servo motor to a certain angle to move the cam and close the gripper, or running through a series of commands to move the stepper motor.

IV. Walkthrough of Key Functionalities

Functions can be run directly from the terminal if that is your preference, but it is often easier to do things through the Pycharm IDE as you can debug, make changes quickly, and there is a nice interface for committing and pushing things to the git repository. You should open Pycharm

from the terminal (Ctrl+Alt+T) and type in “pycharm-community”. This ensures that you have the correct variables, dependencies, etc. whenever you run a program inside of Pycharm. Otherwise, it tends to have some issues running the ROS components correctly.

All code is housed in the mosquito-robot repository, which is on the LCSR Gitlab at (https://git.lcsr.jhu.edu/mosquitoproject/mosquito-robot). This is a private repository, so you will need to request access. You first must create an account by logging in with your JHED credentials. On the Linux machine in the Lab, the repository is located at /home/sanaria/mosq_ws/src/mosquito-robot

A. Camera Calibration (Perform and Check)

This involves a calibration between the camera and tooltip. As such, it will not have to be performed every time you use the robot, or even every time changes are made to the system. Recalibration is required if the gripper is changed (e.g. remounted) or the camera is moved. The calibration is performed using the function ‘run_calibration.py’ that is located in the main directory of mosquito-robot. The comments at the top of this file provide a walkthrough for completing the calibration, which is summarized here.

The calibration is performed on a plane, mapping out a grid of points the robot moves within the image. The user must determine the size of this plane/rectangle over which they want to perform the calibration. This must be done manually by looking at the tooltip in the image. Place a white background in the field of view (e.g. paper). The robot will move at the height (C-axis value) at which it starts, so try to make this as close to the anticipated operating height as possible (e.g. the height of the cartridge/cup surface).

To start the camera:

- First start ROS: open terminal and type “roscore”
- Start the ROS publisher that streams images from the camera. Open another terminal: “python mosq_ws/src/mosquito-robot/cv_camera/scripts/rospub_for_cal.py”
    - Note: this step assumes your terminal is open in the ‘/sanaria/home/’ directory, adjust the address as needed
    - Note: this ‘rospub_for_cal.py’ file turns off auto exposure for the camera and increases the exposure substantially as this has been proven to help the computer vision algorithm that finds the tooltip in the image. Future work should be completed to make the image recognition more robust. Ideally, you could just run the calibration with the standard image publisher that is doing auto exposure: ‘rospub.py’ which is located in the same directory.
- View the image. Open another terminal and type “rqt_image_view”
    - The raw image will be on ROS topic ‘/cv_camera/image_raw’
- You will also want to open another instance of ‘rqt_image_view’ so that you can view a second ROS topic ‘/tooltip_view’ that will begin publishing images only once ‘run_calibration’ is started.

To define the region to perform calibration in:

- Move the robot to the top right corner of the rectangle you want to calibrate in the image (see Fig. 19 for additional clarity). You will have to do this manually e.g. using the ‘jog_robot.py’ function in the main directory of mosquito-robot. You should save the location of the robot at this point and use an absolute move (e.g. in jog_robot) to move here anytime you want to start or restart the calibration.
- Determine the side lengths (in encoder counts) of your rectangle by jogging the robot. These values will be set as “x_range” and “y_range”. “x” is top to bottom of the image, “y” is left to right.
- Determine grid size by setting x_cylces and y_cycles

To run:

- Once the above parameters have been set, jog the robot to the starting position (top right corner of rectangle)
- Run the function. Calibration will proceed automatically - monitor performance via rqt_image_view
- Adjust lighting as needed if the computer vision does not seem to be performing correctly - the small green dot should be on the tooltip in the 'tooltip_view' image

To check the calibration:

- Quantitatively: after the calibration is complete, the function ‘evaluate_calibration.py’ will be run, which will display some statistics and plot the actual and predicted tooltip points
- Qualitatively (e.g. to check if you knocked something out of calibration):
    - One simple solution is to put something distinct (like printed paper) in the background and use ‘move_by_click.py’. Click on some feature in the image and see if the tooltip goes there
    - A more accurate solution is to run a ROS publisher that puts the estimated tooltip position onto the image. To do this, start ROS (‘roscore’) and in another terminal run “python mosq_ws/src/mosquito-robot/cv_camera/scripts/rospub_with_tooltip_calibration.py”. View the image in ‘rqt_image_view’. It will be published to ROS topic ‘cv_camera/image_with_tooltip_est’ and will display a red dot at the estimated location of the tooltip. Visually compare this with the tooltip where you see it in the image.

B. Using RobotMove/DissectorMove packages

To use RobotMove in a function, import the file into your code and instantiate the class, e.g.:

```
from utilities import robot_move
RobotMove = robot_move.RobotMove()  # Create instance of RobotMove class
```

This creation of class instance will commence an initialization (the __init__ function in the package) that includes connection to the Galil controller. At this stage the robot will load in data involved in two safety features of the system:

i. Checking encoder values (encoder value loss due to shutdown detection/prevention):
Any time the robot is appropriately disconnected (i.e. a RobotMove.end_communication() call at the end of a script), a data file called "encoder_at_shutdown.txt" is updated with the current values of the encoders. At startup (at the class instance in the code) the system will ask the robot for the current encoder values and compare them to this file. If they are different, the user is notified, the two sets of values are displayed and the user is asked if it is okay to continue. Note, this discrepancy could be due to:
1) Previous abrupt end of code without closing communication - would be okay to continue
2) Power outage (all zeros) - may consider using reset_encoders_from_file() in this case
3) First 1 occurred, then 2. In this case, would have to re-home

ii. Importing allowed volumes (to prevent collisions):
Allowed volumes (which are explained in detail in the previous section) are saved on the computer and will automatically import at initialization. This effectively ties the allowed volumes to the robot, not the code instance, which is the preferred behavior.

Any functions that explicitly relate to the robot can be carried out via function calls (e.g. RobotMove.move_relative()). The inputs and outputs of these functions are well-documented in comments within the code and in the appendix of this report.

Before running any functions to actuate the gripper (e.g. RobotMove.open_gripper()), the command RobotMove.begin_communication_arduino() must first be run. This is separate from the __init__ function as often in development the robot will be jogged around and initializing the communication with the Arduino includes a small start-up delay that you can skip in those cases.

Any code that uses RobotMove should **always** end with RobotMove.close_communication(). This will ensure that the gripper servo motors powers down and will save the encoder values to a file for potential restoration in the event of a power outage.

DissectorMove was modeled off of the arduino communication in RobotMove, so using it is essentially the same. To use:

```
from utilities import dissector_move

DissectorMove = dissector_move.DissectorMove()
DissectorMove.begin_communication_arduino()
```

Again, always close with DissectorMove.close_communication() so that the motors turn off.

C. Working with Allowed Volumes

These are basically the inverse of obstacles, instead of defining regions where the robot cannot move, the regions in which it can move are provided. This should be a bit “safer” in terms of potential damage to hardware.

Each allowed volume is a rectangular prism in the A,B,C (i.e. cartesian) axes of the robot and is defined by the two corners of this volume that have the lowest and highest values in encoder counts respectively.

Allowed volumes are maintained in a data file on the computer and are loaded into the RobotMove class as a variable when it is initialized in code. This keeps these values consistent between code and operators. The allowed regions will likely need to be changed in the event of hardware changes, either to the gripper, or to the workspace of the robot. This is accomplished through the suite of interface functions in the RobotMove package (where the inputs and outputs are well-documented), which are easily accessible via commenting and uncommenting lines in the manage_allowed_volumes.py code in the main mosquito-robot directory.

Each allowed volume has an associated index, which can be seen along with the defining points of that volume using RobotMove.list_allowed_volumes(). Each time a movement is commanded to the robot, the goal location is checked to see if it is within at least one of these volumes. The check is done in the order of index number, and will stop as soon as an allowed volume has been found (and errors out if none is found).

More volumes can be added using RobotMove.add_allowed_volume(), and can be removed using RobotMove.remove_allowed_volume() either by index number, or via the clear_all optional variable.

It is worth noting that RobotMove can be instantiated with the allowed volumes feature disabled, but this should be only done in rare instances when jogging the robot to determine new bounds for allowed volumes and should be done with **immense caution** as nothing prevents the robot from moving into locations that could damage itself or other equipment. This is completed using an optional boolean limit_to_allowed_volumes in the initial RobotMove call.

D. Loss of Encoder Counts / Homing / Power Outage

As the encoders are incremental, not absolute, anytime the controller loses power, the encoder values will reset to zero at the position the axes are at when the robot controller regains power. Without proper controls, this could cause damage to equipment, and require time wasted homing the robot to restore the correct location values. As such, as described in the “Using the

RobotMove Package” subsection above, there is a data file saved on the computer that contains the encoder counts when the robot was last powered off. These values are compared to the current values at startup and will ask the user to confirm that any differences are acceptable (again, see detailed information in the aforementioned subsection).

If there was a reason to restore encoder values from this file (e.g. the power was turned off, but you know that the communication was shut down with RobotMove.end_communication() before, so the saved encoder values are correct), this can be accomplished via RobotMove.reset_encoders_from_file().

If the correct course of action is to home the robot (e.g. a power outage, but the saved values are suspected to be wrong) this can also be done, but takes more caution and time. Homing takes advantage of the Galil functionality for this task. The axis being homed runs (at whatever currently set speed, which may be quite fast) all the way in the negative direction until it hits a limit switch, then it runs forward until it detects a change in encoder values and sets that as zero. Note especially that the robot will move all the way to the negative end of that axis, and remain there at the end of homing. Care must be taken in the starting location of the robot before each homing move is initiated. Homing of an axis is performed using the RobotMove.home_axis() command. A starting point would be to look at the ‘sanaria_perform_homing.py’ function in the main mosquito-robot directory, but **do not blindly run this script** as the values to bring the robot back to a favorable starting location for each axis before homing will be dependent entirely on the current setup of objects in the robot’s task space and thus must be customized for each homing case.

Remember, the worst case is to cut power to make the robot stop before it hits something, which doesn’t set you back far, especially if you are actively homing at the moment.

E. Running move_mosquito_integrated.py

At the writing of this report, this is the most recent high level control code. While the specifics may become obsolete, some of the initial setup will likely remain applicable to future testing setups.

To start:

- Make sure you opened Pycharm through the terminal "pycharm-community" in order to get all of the correct dependencies!
- To view the behind blades camera:
  Open new terminal and start camera capture program: "guvcview", go to Video Controls and select "USB Microscope" (to save a video, a new path and file name can be created under video-> File)
- To view the behind press camera:
  Open new terminal and start camera capture program: "guvcview", go to Video Controls and select "USB Camera"

- Start ROS: Open terminal: "roscore"
- Start overhead camera image capture
  Open new terminal "python mosq_ws/src/mosquito-robot/cv_camera/scripts/rospub.py"
- To view overhead image:
  Open new terminal: "rqt_image_view"
  Change ROS topic on the rqt image view to "/cv_camera/image_raw"
- Turn on the light sources
- Open Kazam (purple camera icon) to record screen. To stop recording or start if opened before, press camera icon at top right of screen
- Also save the views from the 2 guvcview (hit capture, and press again to stop)

Place the mosquito onto the cup before starting, positioning the proboscis approximately in line with the vertical of the image. Then the program can be run. There is a boolean use_CV that can determine whether the keypoint detection CV method will be used, or whether the user will be presented with an image on which they will click (to simulate the “best computer vision method”). This allows for the mechanical components to be tested separately from the vision system(s). There are also popup confirmations for the user asking if the next step in the process should be completed or skipped. This is helpful for development, when things may not be working right all the time and skipping back to the beginning of the test can save a lot of time. However, once official testing is being conducted, these confirmations can be turned off, allowing for the process to run more autonomously.

V. Deliverables

A. Original Deliverables

Our deliverables were broken into three levels of achievement: a minimum, ideal, and a maximum. Originally our minimum requirement concerned the development and improvement of prototype robotic and dissection systems that had been developed previously by members on this team and our colleagues. The system as it existed had several incomplete or undeveloped subsystems. For example, achieving this deliverable required the design, implementation, and testing of a squeeze mechanism, gland collection device, and body disposal subsystems, all of which had conceptual designs, but no functional prototypes at the start of the project. Our ideal goal was threefold; a written report detailing the mechanical system integration (no use of computer vision), the automated dissection of 50+ mosquitoes, and a written report of the design of a new rotary stage concept. This goal of this deliverable was to produce quantitative results indication the viability of system design concepts as well as shed light on specific issues with component manufacturing or integration. We also planned to develop a clear design concept of a rotary stage design, which would improve efficiency over what was the current dissection concept: a linear stage design. Our maximum deliverable was similar to the ideal except it required us to have completed a written report detailing the system integration with

vision, the automated dissection of 100+ mosquitoes, and a physical prototype of the rotary stage.

At about midway through our project timeline there was a change in priorities at the larger project level. At the suggestion of our mentors, we re-ordered our goals. In order to make a paper deadline we shifted focus to getting the computer vision algorithms integrated with the robot and performing tests with a manual system before attempting mechanical integration with the automated dissection system. This lead to us being in an odd position where we had finished a good percentage of maximum deliverable about a quarter of our ideal and had not achieved our minimum (see Figure 24). By working on the vision system for the paper, our deliverables structure no longer made much sense. This resulted in a restructuring to what we believed was a more reasonable set of deliverables (see Figure 25).

B. New Deliverables

Our new minimum deliverable was to develop a report and videos of the picking and placing of 50 mosquitoes using an integrated vision system without dissection of the mosquitoes. This better aligned with the tasks required of the team early on in the project. Our new ideal deliverable had two parts; first to demonstrate mechanical system integration via the automated dissection of 50 mosquitoes and the second to provide a written report detailing the design concept of the rotary stage dissection system that was deemed acceptable by our mentros. Our new maximum deliverable was an extension of the ideal deliverable. With the first objective being expanded to include gland collection (which is seen as a significant hurtle) and to test with 100+ mosquitoes. In terms of the rotary stage design, the maximum deliverable would have been to fabricate the first generation prototype so that it could be tested.

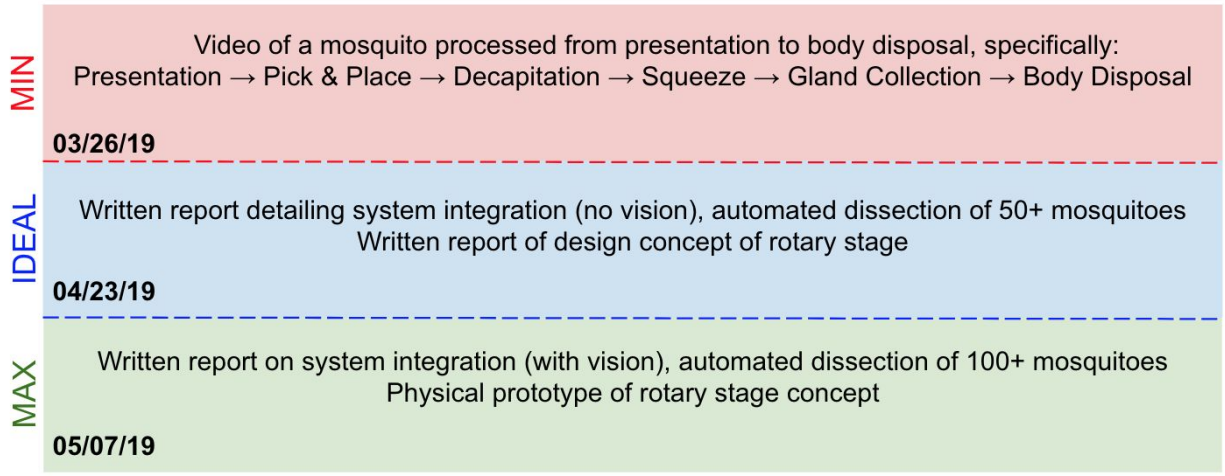


Figure 24. *Table of our original deliverables as defined at the beginning of the project*

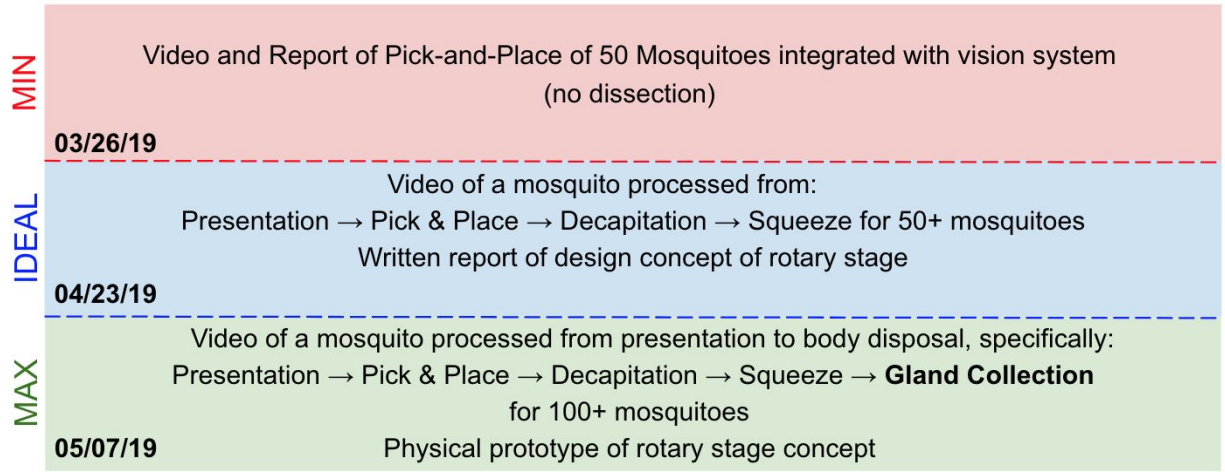


Figure 25. *Table of our redefined and final set of deliverables as defined at the midpoint of our project*

C. Evaluation of Deliverables

Based on our current progress and the redefined set of deliverables we have achieved our minimum and have approached partial completion of our ideal.

Our minimum deliverable of integrating vision with the pick and place robot was achieved. We successfully picked and placed 50 mosquitoes using the robot with the integrated vision system. Video recordings of these trials were taken and the data from these trials were used to report on the accuracy of the system. This report was written for publication in the International Conference on Automation Science and Engineering (CASE). This paper (see Appendix B), which is currently under review, constitutes our minimum deliverable. The key automation task of picking and placing of mosquitoes from a staging apparatus into a dissection assembly was demonstrated in these tests. In the processing of 50 mosquitoes, we demonstrated a 100% grasping accuracy and a 90% accuracy in placing the mosquito with its neck within the blade notches such that its head can be removed [16]. These results were promising since this is a difficult and non-standard pick and place task and the failures provide valuable insight into how to improve the system in future iterations of the system as a whole.

Our ideal deliverable was partially achieved. The ideal deliverable had two parts; the first was a video of a processed mosquito from presentation to squeezing out of the glands for 50 mosquitoes, the second was a written report of design concept of our rotary stage. The first part was not achieved do to some dependencies issues concerning the availability of mosquitoes. Our second part was achieved via a report on the design of a rotary stage to replace the linear stage currently in use. The original design report was adapted and is included here in Section II subsection *C. Design Concept of Rotary Stage*. The final design provided in this section along with the report on the design is what we see as half of our ideal deliverable.

Our maximum deliverable was not achieved. The first part of this two part deliverable was to expand on out ideal by including gland collection and test with 100+ mosquitoes. This was not achieved for several reasons; several weeks without sacrificed mosquitoes, mechanical alignment and precision problems, and a change in gland collection method. We noted in our dependencies that mosquito procurement could pose a risk to meeting our deliverables (see Figure 26), and we did see some weeks without mosquitoes to perform tests due to unforeseen production issues. This lead to several delays in testing of mechanical systems that made it difficult to review and iterate the design to a level where testing could be done of the mechanical systems in a meaningful way. We also underestimated the delays we might encounter due to mechanical design challenges such as tolerancing and stability. The combination of these two issues led to delays in earlier deliverables on which these maximum deliverables depended. Gland collection also was not integrated. This is primarily due to the current project uncertainty in the best way to move forward with this portion of the design. At the start of the semester, the project team was focused on a well-defined method for gland collection, but throughout the semester it has become evident from other tests that we may need to design a new process. As such, this task is less of an implementation task and more of a partnered research task between the teams at Hopkins and Sanaria, which will result in a longer timeline than just implementing an existing, known method. Finally, given the continued design changes in components on all levels in the project, we chose to delay moving forward on building the rotary stage until the rate of design changes of adjacent parts stagnates after their effectiveness has been demonstrated with the linear dissection system.

VI. Dependencies

Overall, we did a good job assessing the dependencies at the start of this project. We identified mosquito procurement as a critical dependency, and despite some issues, we were able to make progress in the weeks without samples. The only major surprise was the change in a long standing assumption of a stable method for gland collection. We originally planned to suction off and collect all of the mosquito exudate, but now, due to purity concerns, a second microdissection step may be required. Additionally, we were not initially aware that mosquitoes had salivary glands that were of similar texture to what was to be anticipated during the production process for only 4 hours after sacrifice. As such, we have now set up a way to procure mosquitoes for testing the day of so we can best test under relevant conditions.

Dependency	Solution	Date Expected	Date Required	Mitigation
Access to shared setup, computer, robot in Robotorium	Coordinate with collaborators	2/26	2/28	Perform testing in off-hours
Access to Dr. Taylor's Lab (Alex)	Ask for Access	2/28	2/28	N/A
Access to mosquitos (weekly basis)	Email colleagues and Sanaria to coordinate pickup	Weekly	Weekly	No testing that week, or unofficial testing with old mosq's or those in ethanol
Interface with computer vision system	Rely on collaborators to continue development	3/15	4/23	Continue to use manual user-click commands
Upstream mosquito staging system	Rely on collaborators to continue development	4/1	4/23	Dissection system can be demonstrated with human-staged mosquitoes
Money for mechanical development (e.g new stage, fabrication costs, etc).	Ask mentors as needed - grant has funding	As needed	As needed	Use what resources are available
Continued functionality of recently re-designed micro gripper	Rely on collaborators to continue ongoing improvements	2/26	2/28	Complete redesign ourselves, possibly adjust project goals

Figure 26. *Table outlining out project dependencies for this project.*

VII. Future Work

Over the past month, we have made it a priority to document and communicate design decisions so that this project can easily be transferred to other students and continued. We will personally assist with the transfer for the month after the semester ends, but will not be continuing on the project in the future due to graduating or transferring to dissertation work. The next steps for this project include the following. We will be conducting formal testing of the dissection system and robot system in the coming weeks, and then a formal test including the vision system(s) should be conducted. Additionally, the rotary stage designed in this project will be manufactured, tested, and iterated as necessary. At this point, it would be valuable to begin efforts to integrate the robot system with the upstage feeding system. To fully automate an end-to-end system would require some consideration to software architecture that may allow for something like a real-time operating system to manage many threads that command each electromechanical system in the assembly.

References

[1] World Health Organization, “World malaria report 2018,” Nov 2018. [Online]. Available: https://www:who:int/malaria/publications/worldmalaria-report-2018/report/en/
[2] M. P. Heron, “Deaths: Leading causes for 2016,” 2018.
[3] World Health Organization, Global technical strategy for malaria 2016-2030, 2015.
[4] S. L. Hoffman, P. F. Billingsley, E. James, A. Richman, M. Loyevsky, T. Li, S. Chakravarty, A. Gunasekera, R. Chattopadhyay, M. Li, et al., “Development of a metabolically active, non-replicating sporozoite vaccine to prevent plasmodium falciparum malaria,” Human vaccines, vol. 6, no. 1, pp. 97–106, 2010.
[5] A. S. Ishizuka, K. E. Lyke, A. DeZure, A. A. Berry, T. L. Richie, F. H. Mendoza, M. E. Enama, I. J. Gordon, L.-J. Chang, U. N. Sarwar, et al., “Protection against malaria at 1 year and immune correlates following PfSPZ vaccination,” Nature medicine, vol. 22, no. 6, p. 614, 2016.
[6] J. E. Epstein, K. M. Paolino, T. L. Richie, M. Sedegah, A. Singer, A. J. Ruben, S. Chakravarty, A. Stafford, R. C. Ruck, A. G. Eappen, et al., “Protection against plasmodium falciparum malaria by PfSPZ vaccine,” JCI insight, vol. 2, no. 1, 2017.
[7] M. S. Sissoko, S. A. Healy, A. Katile, F. Omaswa, I. Zaidi, E. E. Gabriel, B. Kamate, Y. Samake, M. A. Guindo, A. Dolo, et al., “Safety and efficacy of pfspz vaccine against plasmodium falciparum via direct venous inoculation in healthy malaria-exposed adults in mali: a randomised, double-blind phase 1 trial,” The Lancet infectious diseases, vol. 17, no. 5, pp. 498–509, 2017.
[8] K. E. Lyke, A. S. Ishizuka, A. A. Berry, S. Chakravarty, A. DeZure, M. E. Enama, E. R. James, P. F. Billingsley, A. Gunasekera, A. Manoj, et al., “Attenuated pfspz vaccine induces strain-transcending t cells and durable protection against heterologous controlled human malaria infection,” Proceedings of the National Academy of Sciences, vol. 114, no. 10, pp. 2711–2716, 2017.
[9] B. Mordm¨uller, G. Surat, H. Lagler, S. Chakravarty, A. S. Ishizuka, A. Lalremruata, M. Gmeiner, J. J. Campo, M. Esen, A. J. Ruben, et al., “Sterile protection against human malaria by chemoattenuated pfspz vaccine,” Nature, vol. 542, no. 7642, p. 445, 2017.
[10] T. Bousema and C. Drakeley, “Epidemiology and infectivity of plasmodium falciparum and plasmodium vivax gametocytes in relation to malaria control and elimination,” Clinical microbiology reviews, vol. 24, no. 2, pp. 377–410, 2011.
[11] Sanaria Inc. (2014) SporoBot - Build a Robot. Fight Malaria. Save Lives! [Online]. Available: https://www:youtube:com/watch?v=VblazNXcHFg
[12] I. Lapowsky, “The next big thing you missed: This mosquitodissecting, malaria-killing robot needs your help,” Jun 2014. [Online]. Available: https://www:wired:com/2014/06/the-next-big-thing-youmissed-a-crowdfunded-mosquito-dissecting-malaria-killing-robot/
[13] C. Borchers, “Robot may help fight malaria,” May 2014. [Online]. Available: https://www:bostonglobe:com/business/2014/05/07/mosquito-harvest/Qxto58qtpGHhRVfliT6aHI/story:html

[14] R. H. Taylor, A. Canezin, M. Schrum, Iulian Iordachita, G. Chirikjian, M. Laskowski, S. Chakravarty, and S. Hoffman, “Mosquito salivary gland extraction device and methods of use,” U.S. Patent Application US20 170 355 951A1, June 13, 2016.
[15] M. Schrum, A. Canezin, S. Chakravarty, M. Laskowski, S. Comert, Y. Sevimli, G. S. Chirikjian, S. L. Hoffman, and R. H. Taylor, “An efficient production process for extracting salivary glands from mosquitoes,” arXiv:1903.02532, 2019.
[16] H. Phalen, P. Vagdargi, M. Pozin, G. S. Chirikjian, I. Iordachita, and R. H. Taylor, “Mosquito pick-and-place: Automating a key step in PfSPZ-based malaria vaccine production,” Submitted to the 2019 15th IEEE International Conference on Automation Science and Engineering (CASE 2019).

Appendix A. Function Documentation

Table 1: RobotMove Package

Note: most functions will provide a boolean flag output indicating successful completion of the tasks, for simplicity and readability, these outputs are not included in the table below. Axes are referenced throughout as A,B,C,D as Galil references them this way. In the current setup those axes respectively have positive directions to the Left (A), Backwards (B), Down (C), and Clockwise (D) movements.

Function Name	Performed Task	Inputs	Outputs
begin_communication_galil	Initializes communication with the Galil controller. If no input, uses the defaulted address of 192.168.1.1	optional: Ip_address='192.168.1.1' A string of Galil's ip address for ethernet communication	None
begin_communication_arduino	This initializes serial communication with an Arduino microcontroller. After connection a string is also sent that puts the Arduino into a 'connected' mode which turns the servo motor on and opens the gripper.	None	Galil controller class object 'g' used by package to address commands to the controller
set_defaults	Sets default controller gains, speed, acceleration for Galil controller by calling respective functions without optional input.	None	None
close_communication	This closes communication with the Galil controller. As an added convenience feature, before shutdown the current encoder values are written to a file, allowing recover without homing if the power goes out and the incremental encoders lose their positions	None	None
set_controller_gains	Inputs (kp, ki, kd) are tuples or lists of length 4, corresponding to the proportional, integrator, and derivative constants for axes (A,B,C,D) respectively. If you do not wish to change a gain use None as the list element	optional: kp=(6, 6, 6, 6) ki=(0, 0, 0, 0) kd=(64, 64, 64, 64) PID motor control gains for each axis, if left out the default ones shown are used	None
get_controller_gains	Query robot controller for axis PID gains and print to screen	None	kp, ki, kd: lists for length 4 corresponding to [A,B,C,D] axes

set_speed	Input axis_speeds is tuple or list of length 4, corresponding to the speed in encoder counts/sec for each axis [A,B,C,D]. If you do not wish to change a speed use None as the list element	axis_speeds=(25000, 25000, 25000, 25000) Speed in encoder counts per second. One count is approximately 0.5 micron.	None
get_speed	Queries Galil controller for current speed settings	None	speed [A,B,C,D] List of length 4, each element representing speed associated with one axis in encoder counts
set_acceleration	Input axis_speeds is tuple or list of length 4, corresponding to the speed in encoder counts/sec for each axis [A,B,C,D]. If you do not wish to change an acceleration use None as the list element	axis_accels=(256000, 256000, 256000, 256000)	None
get_acceleration	Queries Galil controller for current acceleration settings	None	accel [A,B,C,D] List of length 4, each element representing acceleration associated with one axis in encoder counts / sec^2
move_relative	Instruct the robot to move with some number of encoder counts from current position. If latch is set, function will only return when movement is completed and will check if the final position was reached as commanded. If you do not wish to move an axis use 0 as the list element. NOTE: Rotations can not be commanded with this package. Use Galiltools (or edit) if needed	counts_to_move: tuple or list of length 4, corresponding to the number of encoder counts to move for each axis [A,B,C,D] Optional: latch=True: If true, function will not return until motion is complete	None
move_absolute	Instruct the robot to move to a specific position with tuple or list of length 4, corresponding to the location in encoder counts to move to for each axis [A,B,C,D]. If latch is set, function will only return when movement is completed and will check if the final position was reached as commanded. If you do not wish to move an axis use None as the list element. NOTE: Rotations can not be commanded with this package. Use Galiltools (or edit) if needed	goal: tuple or list of length 4, corresponding to the position in encoder counts where each axis should move to [A,B,C,D] Optional: latch=True If True, function will not return until motion is complete	None
wait_until_stop	When called, this function will not return until it has queried the Galil controller and been told that the specified axes have stopped moving. The default waits on all axes	Optional: axes="ABCD" string of non-separated axis names. e.g. "ABCD", "BC", etc.	none

get_position	Queries Galil controller for current position of the robot	none	position [A,B,C,D] List of length 4, each element representing current position associated with each axis in encoder counts
get_velocity	Queries Galil controller for velocity of each axis of the robot	none	velocity [A,B,C,D] List of length 4, each element representing current velocity associated with each axis in encoder counts / sec
open_gripper	Send the arduino a serial command to move the servo and open the gripper	none	none
close_gripper	Send the arduino a serial command to move the servo and close the gripper	none	none
get_gripper_state	Tells what state the program believes the gripper to be in (i.e. what was the most recent instruction	none	gripper_state Integer ‘boolean’, 1 for open, 0 for closed
home_axis	Starts a homing routine on specified axis. Will move the specified axes in the negative direction until the limit switch, advance forward to an encoder pulse, and set that position as 0.	axis String to specify axes with its character name as a string (e.g. "B", "C" or "D")	none
set_position	Input new position values for current position with. If you do not wish to change the value of an axis, set list element to None.	Positions Tuple or list of length 4 that defines position for each axis [A,B,C,D]. Each element integers or None	none
read_allowed_volumes	Read the allowed volumes from local file (typically used during init). If no file found, initialize allowed volumes list as empty.	filename String containing address to file where allowed volumes are stored on the local computer	allowed_volumes Numpy array containing all allowed volumes that were stored in the file (or an empty array if no file)
add_allowed_volume	Add an allowed volume in which the robot is allowed to be commanded. This is defined through the two extreme corners of a rectangular prism in the robot’s workspace. For now, rotation is not considered as it is disallowed in this package. These values will be added to the class variable that keeps	low_bound, high_bound Lists of length 3 comprised of integers. Values given as [x,y,z], corresponding to the values of points at the corner of a 3D volume in which the robot is	none

	track of allowed volumes for checks before move commands, as well as to the file cataloging allowed volumes so that it will remain in the future.	allowed to move. The values correspond to the x,y,z (or a,b,c) axes respectively. The values of all elements in low_bound must be strictly less than those in high_bound. All values are in encoder counts of the robot.	
remove_allowed_volume	Removes one or more allowed volumes for the system, both in the local class variables and in the file from which allowed volumes are read in at startup. List the volume indices in a list [i,j,k]. Try using 'list_allowed_volumes' to see which ones to remove. Set clear_all flag to true to completely reset volumes.	remove_indices List containing integers corresponding to the indices of the allowed volume you want to remove Optional: clear_all=False Setting this flag will remove all volumes, regardless of the indices supplied	none
list_allowed_volumes	Prints a viewing-friendly list of all allowed volumes to the screen. Can be used along with 'remove_allowed_volume' to find candidate volumes for removal	none	Prints a user-friendly list of allowed volumes with associated index number
check_if_allowed	Input is a commanded for cartesian motion (rotational moves not allowed currently), check to see if that position would like in an allowed region. If so, return, else quit.	command position list of integers of length 3 in [x,y,z] (or [A,B,C]) coordinates	none
save_encoder_vals	Saves current encoder values to file. When run at shutdown prevents re-homing in event of controller losing power	none	Writes current encoder values to a file on the local computer
read_encoder_vals_last_communication_end	Reads encoder values from data file saved to with save_encoder_vals	none	last_position List of length 4 corresponding to encoder counts for [A,B,C,D] axes
reset_encoders_from_file	Resets encoder values from the file saved in save_encoder_values. To be used in event of controller loss of power	none	none
gal_out_to_list	Helper function: Converts output string for Galil controller to Python list	galil_output String from controller	list_output List of integers
list_to_galil_input	Helper function: Turn list into galil controller-friendly input	galil_command_str String indicating Galil command list_input List of integers	galil_input String for controller

Table 2: DissectorMove Package

Note: most functions will provide a boolean flag output indicating successful completion of the tasks, for simplicity and readability, these outputs are not included in the table below

Function Name	Performed Task	Inputs	Outputs
begin_communication_arduino	This initializes serial communication with an Arduino microcontroller. If the Arduino has not been hard reset (e.g. power off, disconnected from computer, new command flashed), then the current slot should be read in,preventing the stage from needing to be re-homed	Optional: port='/dev/ttyACM1' Specify the port the Arduino is on.	none
move_to_home	Sends serial command to have the linear stage home itself and advance to the first active slot , optional (defaulted) latch to keep function from returning until action is complete	Optional: latch=1 When set does not return until Arduino is finished action	none
move_forward_one_slot	Sends serial command to move one slot forward, optional (defaulted) latch to keep function from returning until action is complete	Optional: latch=1 When set does not return until Arduino is finished action	none
move_backward_one_slot	Sends serial command to move one slot in reverse, optional (defaulted) latch to keep function from returning until action is complete	Optional: latch=1 When set does not return until Arduino is finished action	none
move_to_slot	Move to a specific slot number - automatically latched so that it can do these in sequence	target_slot Integer specifying cartridge slot to move to	none
actuate_cutter	Sends serial command to activate cutter, optional (defaulted) latch to keep function from returning until action is complete	Optional: latch=1 When set does not return until Arduino is finished action	none
actuate_press	Sends serial command to activate press, optional (defaulted) latch to keep function from returning until action is complete	Optional: latch=1 When set does not return until Arduino is finished action	none
close_communication	Closes communication to the Arduino which powers down the servo motors, stopping chattering. Also allows the Arduino to temporarily save slot value in memory, so that if you reconnect	none	none

	without a hard reset on power, you do not have to re-home the stage		
__wait_until_complete	Delays function progress until Arduino is finished, optional return of printed line	none	output The string sent by the Arduino

Table 3: Important Files and Scripts

Name	Relevant Contents / Tasks
utilities/image_coord_cursor_overhead.py	When called, will pull current image from ROS topic on which overhead camera image is streamed. User clicks anywhere on this image and the pixel value is provided. This allows for the user to take the place of the computer vision system for unit testing
utilities/bernstein_polynomial.py	Written by previous students who worked on project. We treat as a "black box" mostly. We did confirm that it is implementing Bernstein polynomial fitting correctly. We made some improvements, such as converting data type to numpy arrays and improving coefficient storage. get_est is an important function to go from camera to robot coordinates get_bern_cof takes calibration results and generates the Bernstein coefficients
utilities/gclib.py	Existing python library for Galil controller communication that RobotMove wraps
utilities/keypoint_vision.py	Contains functions written primarily by Hongtao Wu to implement a neural network that identifies key points on mosquito anatomy
utilities/Centroid_subscriber.py utilities/dl_inference.m utilities/dl_subscriber.py	Several functions written by Prasad Vagdargi to implement neural network to do identification of mosquito in image and segment parts
arduino_code/arduino_for_prototype.ino	The code that is flashed to the Arduino for the dissector system
arduino_code/servo_gripper.ino	The code that is flashed to the Arduino for the gripper
cv_camera/scripts/rospub.py	Streams images to rostopic from overhead camera
cv_camera/scripts/autodetector.py	Streams images to rostopic from overhead camera AND automatically detects a mosquito in the image and publishes its centroid location
cv_camera/scripts/rospub_for_cal.py	Streams images to rostopic from overhead camera, but has a high exposure time, washing out potential false positives in calibration images
evaluate_calibration.py	Shows graph of actual location of robot during calibration and predicted locations using the Bernstein polynomial fitting. Supplies statistics of this fitting to those calibration points
jog_dissector.py	Quick code to run functions in DissectorMove package

jog_robot.py	Quick code to run functions in MoveRobot package, especially to move the robot around or open/close the gripper
manage_allowed_volumes.py	Quick code to manage the allowed volumes in the RobotMove package. Easy to add, remove, etc. from here
move_by_click.py	Image is displayed to the screen, user clicks on it and the robot moves there. Good for quickly moving the robot somewhere or qualitatively checking calibration
move_mosquito_CASE2019.py	The automatic mosquito pick-and-place algorithm used for the experiments which were submitted to CASE 2019. This used the segmentation neural network. The sAMMS was used mechanically.
move_mosquito_auto_keypoint.py	Automatic mosquito pick-and-place algorithm using keypoint detection neural network. The sAMMS was used mechanically.
move_mosquito_integrated.py	Either autonomous or semi-autonomous mosquito pick-and-place algorithm (using keypoint computer vision or user clicks). The linear dissection system was integrated and is controlled by this code.
recal_from_file.py	Recomputes Bernstein coefficients from data files collected during calibration. Useful if you had an old calibration saved and wanted to reuse or there was some error
run_calibration.py	Move robot to starting position and specify how far you want the robot to move in each direction and the grid density. Then run the function. Uses a computer vision method to find the tooltip in each image.
sanaria_perform_homing.py	Performs homing using RobotMove package, but should move the robot to appropriate locations after each sequential move to prevent collision. NOTE: will need to be edited for changes to robot workspace. It may be useful to perform this line by line and get the robot in the correct starting position for each axis homing. See the relevant section of this report.

Appendix B. CASE 2019 Manuscript

This manuscript was developed by members of our group based on work done during this project. It is attached below as a reference for more detail on items discussed in the main body of this report.

Mosquito Pick-and-Place: Automating a Key Step in PfSPZ-based Malaria Vaccine Production

Henry Phalen, Prasad Vagdargi, Michael Pozin, Sumana Chakravarty, Gregory S. Chirikjian, *Fellow, IEEE*, Iulian Iordachita, *Senior Member, IEEE* and Russell H. Taylor, *Life Fellow, IEEE*

***Abstract*— The treatment of malaria is a global health challenge that stands to benefit from the widespread introduction of a vaccine for the disease. A method has been developed to create a live organism vaccine using the sporozoites (SPZ) of the parasite *Plasmodium falciparum* (Pf), which are concentrated in the salivary glands of infected mosquitoes. Current manual dissection methods to obtain these PfSPZ are not optimally efficient for large-scale vaccine production. We demonstrate the automation of a key step in this production process, the picking and placing of mosquitoes from a staging apparatus into a dissection assembly. This unit test of a robotic mosquito pick-and-place system is performed using a custom-designed micro-gripper attached to a four degree of freedom (4-DOF) robot under the guidance of a computer vision system. Mosquitoes are autonomously grasped from a mesh platform and pulled to a pair of notched dissection blades to remove the head of the mosquito, allowing access to the salivary glands. Placement into these blades is adapted based on output from computer vision to accommodate for the unique anatomy and orientation of each grasped mosquito. In this pilot test of the system on 50 mosquitoes, we demonstrate a 100% grasping accuracy and a 90% accuracy in placing the mosquito with its neck within the blade notches such that its head can be removed. This is a promising result for this difficult and non-standard pick-and-place task. Analysis of the failure cases provides insights for improvements to be implemented as we integrate this robotic pick-and-place system into a larger automated mosquito dissection system under development.**

I. INTRODUCTION

Malaria presents a tremendous public health burden. The World Health Organization estimates 219 million individuals worldwide were infected with the disease in 2017 and ranked it among the top 20 leading causes of death among both adults and infants in 2016 [1], [2]. With increasing drug and insecticide resistance, it has become ever more difficult for current treatments to maintain efficacy in reducing the prevalence of malaria worldwide [3]. Development of malarial vaccines present a promising way forward in the global effort for malaria eradication [3]. Progress has been made in the development of the Sanaria *Plasmodium falciparum* sporozoite-based vaccine (Sanaria® PfSPZ Vaccine), an effective vaccine manufactured from PfSPZ extracted from the salivary glands of female *Anopheles* mosquitoes [4]–[9]. Such a vaccine may reduce the burden of the disease by providing immunity against Pf, the most common malarial parasite, which was estimated to account for greater than 95% of deaths caused by malaria in 2017 [1], [10].

The process of vaccine production requires salivary gland dissection and to date has only been demonstrated with training-intensive manual or semi-automated processes, presenting a major bottleneck in the scalability of this vaccine. In traditional manual methods, technicians are presented with freshly-sacrificed mosquitoes and process them one at a time, removing the mosquito's head with a needle under microscope and squeezing out a volume of exudate that includes the PfSPZ-laden salivary glands. The exudate from mosquitoes is collected and processed for the isolation of PfSPZ.

The automation of salivary gland harvesting from *in vivo* mosquitoes has been attempted in the past [11]–[13]. However, no literature supports the success of any such process at this time. A semi-automated mosquito micro-dissection system (sAMMS) has been developed and investigated within our research group [14], [15]. In the sAMMS process, a human technician uses micro-forceps to sort mosquitoes into cartridges such that their necks extend between cutter blades. Then, the blades are actuated to cut off all the heads, and a comb-like squeezing device is used to extrude all the exudate, which is collected via a suction device. Early experience has shown that this device roughly doubles the throughput of purely manual dissection and reduces training time to reach peak performance from 39 to 1.5 weeks [15].

While a demonstrable improvement over manual methods, the sAMMS device was developed only as a first step towards a fully automated dissection system to enable large-scale production of enough vaccine for world-wide vaccination efforts. One major bottleneck in the transition to a fully automated system is the visual perception and physical precision it takes to recognize a mosquito, analyze it, and best align it so that the head can be removed. We report our work to overcome these challenges through the development of a vision-guided pick-and-place robotic system for loading mosquitoes into the sAAMS device. As our research group works toward development of a fully automated mosquito salivary gland dissection system, the demonstration of a robust pick-and-place apparatus is a key milestone in realization of that goal.

*This work was supported by NIH SBIR grant 1R44AI134500-01 in collaboration with Sanaria, Inc. Rockville, MD, USA. Additionally, H. Phalen is supported by the NSF Graduate Research Fellowship under Grant No. 1746891. Information contained herein is both confidential and proprietary to Sanaria Inc. and JHU. This document is being provided for conference review purposes only, pending filing of patents prior to the conference.

H. Phalen, P. Vagdargi, M. Pozin, G. S. Chirikjian, I. Iordachita, and R. H. Taylor are with The Laboratory for Computational Sensing and Robotics (LCSR) at the Johns Hopkins University, Baltimore MD, USA. S. Chakravarty is with Sanaria, Inc. Rockville, MD, USA. H. Phalen is the corresponding author, henry.phalen@jhu.edu

Manuscript 475 submitted to 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Received March 19, 2019.

II. SYSTEM DESIGN CONCEPT

A. System Overview

The robotic pick-and-place system described in this paper will function as a subsystem within a fully-automated mosquito dissection system. This larger system will ultimately take freshly-sacrificed mosquitoes suspended in water and output a collection of mosquito exudate including PfSPZ-laden salivary glands. Our concept of this dissection system is provided in Fig. 1. We briefly describe this system to clarify the context of the robotic pick-and-place subsystem, which will be the middle of three primary system components.

First, a staging apparatus will separate mosquitoes and present them one at a time to the robot. Freshly-sacrificed mosquitoes sit in a basin of solution beneath the system. A spinning rotor in the basin creates a vortex that will carry mosquitoes in solution to the top of a separation cone. This cone has channels in one sector down which water will flow onto a ring of orientable mesh-bottomed cups. This ring will rotationally index around the cone so that, by controlling the vortex speed and concentration of mosquitoes in the basin, the cups will on average have one mosquito on them once they pass beyond the sector of the cone with channels. At an index beyond the channel, a camera will image a single cup and a computer vision algorithm will determine if a mosquito is present. If so, at the next indexed position, the cup will be rotated to orient the mosquito so that the mosqutio's proboscis will point radially outward from the ring. Finally, the ring will be rotated to an index that a sits parallel to a linear stage that will comprise the third subsystem, a dissection assembly line. The development of the staging apparatus is described in detail in [16].

The pick-and-place robot will be positioned on the other side of the linear stage and will reach over to the cup, grasp the mosquito by its proboscis and drag it onto a sAMMS-resembling cartridge attached to a linear stage. Similar to how a human technician would use the sAMMS, the robot will drag the mosquito into a slot and place the mosquito's neck into notches cut in two parallel dissection blades. An overhead camera will be used to provide computer vision feedback of this process. The blades will be actuated, cutting the head. After disposing of the mosquito's head, the robot will return to the ring which will have rotated to present a new mosquito. The linear stage will index laterally immediately after the mosquito is cut. As additional mosquitoes bodies are positioned on the cartridge, the linear stage will translate and expose mosquitoes to several stations at which the exudate can be squeezed out and salivary glands collected. Work on the dissection assembly line is currently ongoing within our research group.

B. Requirements

We focus here on the robotic pick-and-place system, along with its difficult and important task of picking up mosquitoes presented on a mesh surface, and precisely placing them so that only the neck lies within the blades. In order to extract

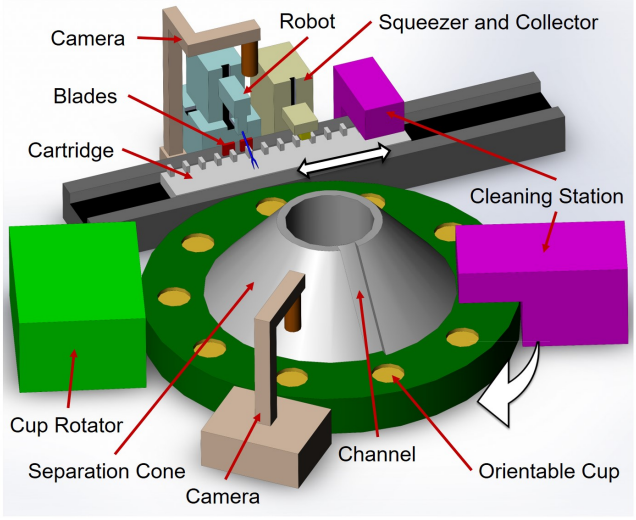


Fig. 1. Concept image for automated mosquito dissection system.

the salivary glands of the mosquito, the dissection point has to be precisely at the intersection of the head and thorax. If the mosquito is not placed far enough into the blade, the cut will occur on the head, leaving no passage for the exudate to be squeezed out, effectively wasting the mosquito and PfSPZ living within. Because the salivary glands are located just behind the mosquito's head, placing the mosquito too far into the blade would result in some of the gland being lost in the cut, also decreasing PfSPZ yield. The mosquito neck is approximately 0.3 mm in length, and the blades each have a thickness of 0.002 inches (0.051 mm), leaving only about 200 micrometers for error. Moreover, grasping must occur only on the proboscis to prevent any damage to the body that might ruin the salivary glands or create an alternative opening for exudate to squeeze out of. The proboscis is on average 2.0 mm long, with a diameter of approximately 0.1 mm.

In addition to size challenges, this procedure presents multiple difficulties not typically faced in a standard pick-and-place procedure. One of the main challenges is the mosquito-to-mosquito variation. Some of this is anatomical in origin. Each mosquito varies somewhat in size and is not axis-symmetric, meaning the alignment of the neck relative to the body depends on which side the mosquito body lies on. Further, mosquitoes are very flexible. By grabbing and pulling the mosquito from the proboscis, the body tends to straighten out in time for placement, but first, the mosquito has to be identified and grasped from all variety of twisted, compressed, or otherwise contorted orientations. While upstream processes are expected to align the mosquito's proboscis within 15 degrees of an ideal orientation for the robot to grab it, the mosquito can still be located anywhere on the cup and must be grasped accordingly. Because of its length, the proboscis can still be grabbed even if there is some error in the robot positioning, or computer vision targeting. However, this, combined with the general variability in proboscis lengths, means that the

Manuscript 475 submitted to 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Received March 19, 2019.

offset between the robot's grasp point and the mosquito's neck is not consistent trial-to-trial. As such, these challenges necessitate adaptive automation. It is not enough to program a sequence of movements, but rather for each mosquito, the robot must move to a unique location, grasp, and then the determine exactly where to move for placement based on visualization of the mosquito's anatomy and the grasping location.

C. Experimental Setup

The robot used in this procedure is a 4-DOF, linear stage robot by New England Affiliated Technologies, Lawrence, MA (Fig. 2). A dual-axis X-Y table is used as the base for the robot, onto which a Z axis is mounted orthogonally (NEAT: XYR-6060 and NEAT: LM-400 respectively). The robot also has a rotary axis which is not used in this study. Each axis is driven by a 12V DC servo motor, with a leadscrew, has a travel of 100 mm, and is coupled with an incremental encoder. The positioning resolution of these axes was measured with a dial indicator to be approximately 10 micrometers. The entire assembly is mounted to an optical table. Robot motion is driven by a Galil controller (DMC-4143), interfaced to a Linux computer by ethernet connection. Attached to the robot is a custom-designed micro-gripper mechanism visualized in Fig. 3. A cam mechanism attached to a HexTronik HXT900 servo motor drives the rail of a linear guide within its carriage, causing the tooltip to open and close. The tooltip of the micro-gripper is adapted from an Alcon Grieshaber retinal surgical forceps. Movement of the linear guide rail extends or retracts a sleeve over a normally-open gripper jaws. The micro-gripper is controlled by sending position commands to the servo motor via USB serial communication from the computer to an Arduino Uno microprocessor.

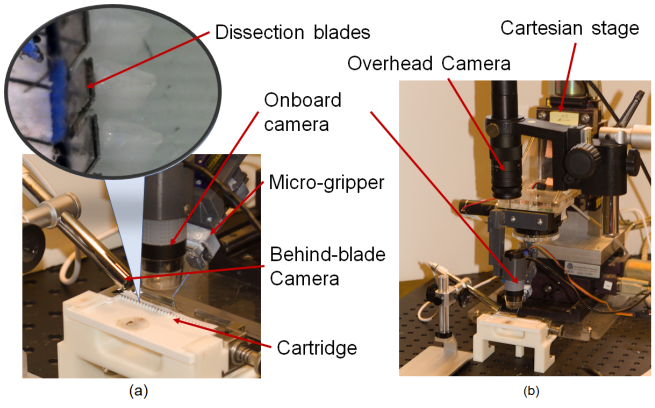


Fig. 2. Experimental Setup. A close-up image of dissection blades and cartridge is inset.

Mosquitoes are staged for dissection on a modified sAMMS device that is also mounted to an optical table. The sAMMS cartridge is modified to have a hole 23 mm away from the blades in which is placed a 20 mm diameter cup that matches the one used in the upstream staging apparatus. This cup is covered with a nylon 750 micrometer mesh that

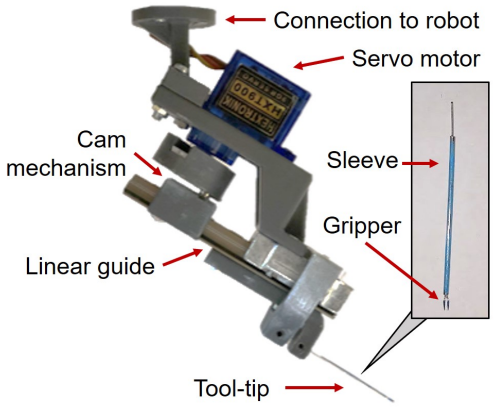


Fig. 3. Custom-designed micro-gripper used to grasp mosquitoes.

is used for water drainage in that apparatus. The mosquito is dragged into a slot in the sAMMS cartridge and placed into the sAMMS blades. These are two 0.002 inches (0.051 mm) thick stainless steel blades with 0.5 mm wide by 1.0 mm deep notches cut in them to match the midpoint of the slots. The closest blade to the cartridge is stationary while the further blade can be manually actuated by pressing a button on the side of the device. This action causes the mosquito neck to be caught between the two blades and cut.

The setup also includes three cameras (Fig. 2). An overhead microscopic camera (OptixCam Summit D3K2-5) with an Omano OM-10K zoom lens is used to capture a complete view of the workspace and is used by the computer vision to identify a mosquito's presence and general location. A second camera (Plugable USB Microscope Camera), is mounted on the robot and is used to identify the location of the mosquito's body parts for accurate picking and placing. We refer to these as the overhead and onboard cameras respectively. A third camera (Opti-Tekscope USB Microscope Camera) is placed to the side and rear of the setup so that it's visual field is in line with the blades. This camera is not necessary for system operation and is only used to visualize placement so the tester can determine if a trial was successful or not.

The automated procedure uses the overhead and onboard cameras to guide the robot's motion. The procedure consists of three stages. In the first stage, an image of the entire workspace is captured using the overhead camera. This image is converted to HSV space, and the mosquito is segmented out. Next, a bounding box is fit to this region and a weighted centroid is calculated for the mosquito, as shown in Fig. 4(a). Using the calibration method described in Section II D, the tooltip is moved to mosquito. At this position, the onboard camera is able to capture a much more zoomed in image with more features and details.

In the second stage, the onboard camera captures a detailed image of the mosquito and identifies the proboscis in the image shown in Fig. 4(b). The centroid of the proboscis is used as the grasp location for the mosquito. Using the calibration procedure in Section II D, the tooltip is moved

Manuscript 475 submitted to 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Received March 19, 2019.

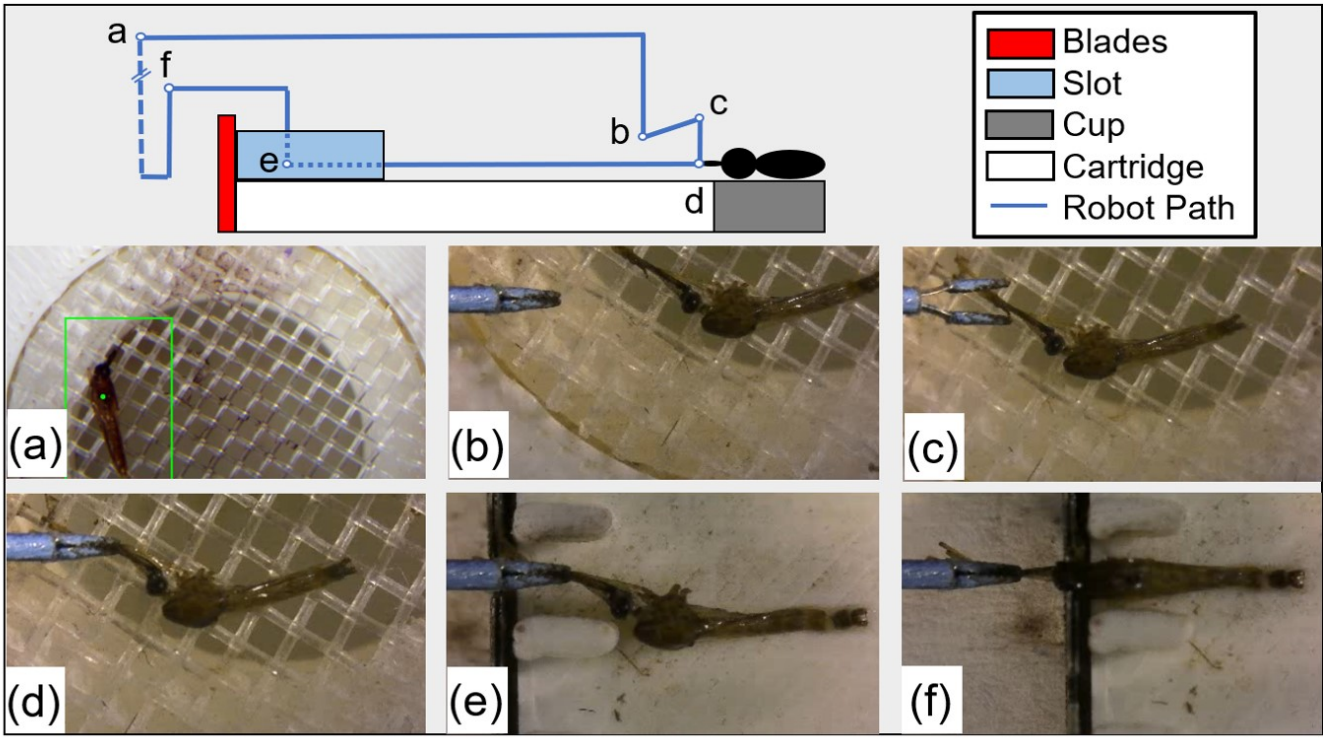


Fig. 4. Side view of robot path and related representative images captured by the vision system. (a) Image captured from overhead camera showing bounding box of detected mosquito. (b) Image captured from onboard camera to determine proboscis centroid. (c) Image captured from onboard camera before grasping. (d) Image captured from onboard camera immediately after grasping. (e) Image of the mosquito taken used to calculate head-to-tooltip offset. (f) Image after aligning the mosquito neck with the blades.

to the grasp location 4(c) and the mosquito is grasped by its proboscis 4(d). Finally, the robot drags the mosquito to an empty slot on the cartridge near the blades.

In the third stage, the onboard camera captures a final image shown in Fig. 4(e) with the tooltip in view to detect the mosqutio head-to-tooltip offset. The robot uses this offset value to position the mosquito with its neck between the dissection blades (Fig. 4(f)). Our group is also investigating the use of a second computer vision processing algorithm [17], which was not used in this study.

D. Calibration

To relate the robot and camera coordinate systems, we use a two stage calibration process. In the first stage, the tooltip of the micro-gripper is located in the overhead camera frame. The tool is segmented from the background in HSV space using Otsu's binarization [18], and contour identification is used to detect the tool. The lowest point of the tool contour is then used as the tooltip. In the second stage of calibration, the robot is moved through the camera space across a grid of points. The tooltip is detected and recorded at each position. The resulting grid of points from both coordinate systems are used as the inputs for a Bernstein polynomial fitting routine as performed in [19]. This routine fits two fourth degree polynomials, which creates a bijective map from the camera coordinates to the robot's encoder coordinates. Such a polynomial fitting method also compensates for radial and aspherical lens distortions.

The tooltip does not move with respect to the overhead camera, so the polynomial fitting method described above could not be used to calibrate this camera. Instead, a pre-calibrated grid of a resolution 5mm x 5mm was placed in the background. The robot was then moved by a known distance along each axis, and images were captured before and after motion. The pixel motion of the grid was calibrated to the corresponding change in robot encoder counts.

The location of the cartridge and blades in robot coordinates were determined using a shim. The robot was slowly advanced until the robot held the shim firmly to the surface of interest and encoder counts at this location were used as a reference.

III. EXPERIMENTAL METHODS

A. Study Design

Testing was performed to investigate the efficacy of the designed system to pick up a mosquito and place that mosquito within blades that can remove the insect's head. The process was performed on 50 non-infected *Anopheles* mosquitoes. Prior to testing, the mosquitoes were kept in an airtight, refrigerated container of phosphate-buffered saline solution (PBS) following sacrifice one day prior. Functioning as a unit test for this subsystem within the eventual automated mosquito dissection system, only the grasping and subsequent positioning of the mosquitoes by the robot were considered for trial success or failure. All actions of the

Manuscript 475 submitted to 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Received March 19, 2019.

system during the test were performed autonomously with feedback from computer vision.

B. Pick Procedure

The experimental procedure is demonstrated in Fig. 4. A mosquito is removed from the PBS solution by its proboscis with tweezers and placed anywhere on a circular mesh cup of radius 10 mm with its center placed 23 mm away from the blades as measured on the central axis of the cartridge slot. The mosquito was placed so that the proboscis was positioned forward toward the blades, pointing within 15 degrees this line. One such placement is shown in Fig. 4(a). These conditions were chosen to mimic those expected from the upstream mosquito-staging apparatus that this process will later be integrated with. To further match the expected results of this upstream process, no further attempt at standardization of mosquito starting position were made (e.g. what side the mosquito was lying on, relative straightness of legs). The micro-gripper tooltip begins the trial at a location away from the cup and 3.5 mm above the cartridge surface.

A bounding box around the mosquito is identified by computer vision in an image from the overhead camera, and the robot moves to a point 5.0 mm in front of the centroid of that region (Fig. 4(b)). This brings the mosquito into view of the onboard camera without placing the gripper over top of the mosquito body. By lowering 3.0 mm toward the mesh surface, the mosquito is brought into focus. The centroid of the proboscis region is identified and the robot moves the gripper to a location 2.0 mm above this this point (Fig 4(c)), and then drops down to the mesh surface and the gripper is closed to grab the proboscis (Fig 4(d)).

The robot lifts up 0.8 mm and drags the mosquito to a position 1.5 mm from the blades (Fig. 4(e)). Here, an image from the onboard camera is again analyzed by the computer vision system. This task serves two functions, to confirm successful grasping of the mosquito, and to determine more accurately where on the proboscis the gripper has grabbed. The trial is considered a successful demonstration of grasping if the mosquito is visualized as grasped within the micro-gripper at this point.

C. Place Procedure

The vision system provides the location of the proximal end of the proboscis, where it attaches to the mosquito's head. This location is transformed into robot coordinates and a head-to-tooltip offset is determined by subtracting it from the current encoder values. Only the offset in line with the cartridge grooves (a horizontal offset in Fig. 4(e)) is considered. The robot then executes another set of programmed movements. The robot raises the mosquito head 1.3 mm and moves forward a nominal distance to clear the blades plus the offset, such that the mosquito's neck should be right above the blades (Fig. 4(f)). Then the tooltip moves down 3.0 mm, placing the neck within the notch of the blades if properly aligned. At this point, another subsystem of the automated mosquito dissection system would actuate the blades to cut the head and further process the mosquito. In this unit test, the blade is manually actuated. The test is considered a successful placement if the mosquito's neck is placed into the notch of the dissection blades such that the head could be removed. As a final step of the process, the robot pulls away from the blade, moving the head, if still in its grasp, to a location where it can be blown or rinsed off. Video footage from all three cameras is recorded throughout and saved for analysis. The robot axes are programmed to execute all moves at a speed of 12.5 mm/s except the final drop to lower the neck between the blades at which the robot is programmed to move at 2.5 mm/s.

IV. RESULTS

Throughout the experiment, there were no issues moving to a mosquito's location, grasping it by the proboscis, or dragging it on the surface of the cartridge. All 50 (100%) of the mosquitoes were observed with the proboscis grasped by the micro-gripper during the second vision check (Fig. 4(e)). Of these 50 mosquitoes, 45 (90%) were placed such that their necks were aligned correctly within the blades. This was determined by review of closeup video at the blades. An example of a mosquito being accurately placed is provided in Fig. 5 (a).

The five mosquitoes that were not accurately placed exhibited similar behavior, flipping over the blades when pulled down by the robot. This action is demonstrated in Fig. 5 (b). In these cases the mosquitoes appear to collide with either the slot walls or the blades. That collision point acts as fulcrum, causing the downward motion of the robot to flip them over the blades, rather than pull the neck into the notches. We were unable to correlate this behavior with any other variable including initial mosquito orientation, grasp location of the proboscis, trial number, or a qualitative assessment of the computer vision's head-to-gripper offset estimation.

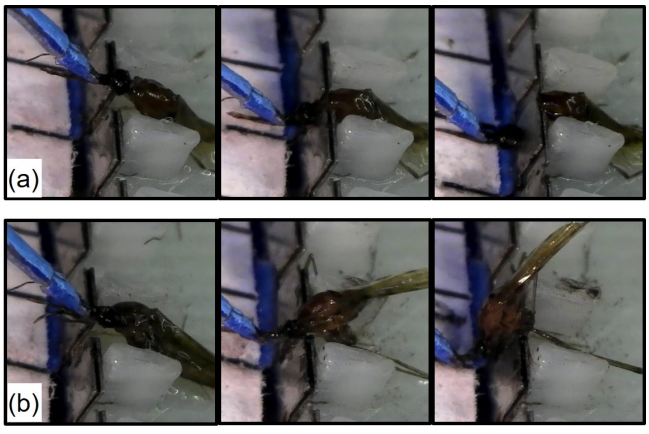


Fig. 5. A demonstration of mosquito placement. (a) A mosquito being accurately placed with neck between the blades. (b) A mosquito being flipped over the blades during attempted placement.

V. DISCUSSION

The robotic pick-and-place subsystem demonstrated highly successful results in this unit test. With no failures in grasping or moving the mosquito, the micro-gripper is shown

Manuscript 475 submitted to 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Received March 19, 2019.

to be adequately designed for the task. In order to both pick and place any single mosquito, the system was required to navigate to locations that were not explicitly programmed. The ability of the system to achieve these promising results indicates that the computer vision system was effective at providing appropriate adaptations to robot movement.

Although we are still working to improve the system, the 90% success rate from this initial pilot study is very encouraging, especially considering the challenges presented by this non-standard pick-and-place task. It is not surprising that the placing task would prove more difficult than grasping the mosquitoes as it requires more accuracy. In order to move the mosquito, the robot can grasp anywhere on the length of the proboscis. Placing the mosquito's neck between the blades requires more precision and any inconsistencies in grasping, deformation, and anatomy must be accommodated in this step. Using a head-to-tooltip offset determined by the vision system worked well and we will continue to enhance the performance through improvements to the vision algorithm at this step.

In the few cases where adequate placement was not achieved, the mosquitoes were observed to flip over the blades about a contact point with either the blades or the cartridge. This behavior occurred both when the neck appeared to be misaligned with the blade notches as well as in cases when the alignment appeared adequate. When there was neck misalignment, either the head or body of the mosquito, which are wider than the notch within the blades, contacted the top of the blades and caused the mosquito to flip over when the robot pulled the proboscis downward. Our work to better determine the tooltip-to-head offset should improve the accuracy of alignment, and we will also investigate if adjusting the angle from which the images are taken can improve our estimation. As the robot holds the proboscis above the cartridge surface, its length is foreshortened in the top-down view provided by the onboard camera. A better estimation of the offset may be obtained geometrically or from a side-view camera where the proboscis profile should not be distorted. We will also target further improvement through mechanical changes to the blade and cartridge geometries to better guide the mosquito neck into position even in cases of small errors in robot positioning. These modifications should also address the situations in which alignment appeared adequate by video observation, but a flip still occurred.

VI. CONCLUSION

We have presented the design and implementation of a robotic system for performing the challenging and non-standard pick-and-place task involved in mosquito dissection. Successful demonstration of this process represents a major milestone in our effort to automate the malaria vaccine production process. The micro-gripper performed with high accuracy and consistency and the system's performance indicates reliable vision and calibration techniques. Our methods proved to adapt well to anatomical and positional differences amongst the tested mosquitoes. These results will help to inform the further design of this subsystem, its companions, and their interfaces, ultimately contributing to an automated mosquito dissection system to facilitate the scalable production of PfSPZ-based malaria vaccines.

ACKNOWLEDGMENT

This work was supported by NIH SBIR grant 1R44AI134500-01. Additionally, H. Phalen is supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1746891. The authors would like to thank Amrita Krishnaraj and Jiteng Mu for their preliminary efforts.

REFERENCES

[1] World Health Organization, "World malaria report 2018," Nov 2018. [Online]. Available: https://www.who.int/malaria/publications/world-malaria-report-2018/report/en/

[2] M. P. Heron, "Deaths: Leading causes for 2016," 2018.

[3] World Health Organization, *Global technical strategy for malaria 2016-2030*, 2015.

[4] S. L. Hoffman, P. F. Billingsley, E. James, A. Richman, M. Loyevsky, T. Li, S. Chakravarty, A. Gunasekera, R. Chattopadhyay, M. Li, *et al.*, "Development of a metabolically active, non-replicating sporozoite vaccine to prevent plasmodium falciparum malaria," *Human vaccines*, vol. 6, no. 1, pp. 97–106, 2010.

[5] A. S. Ishizuka, K. E. Lyke, A. DeZure, A. A. Berry, T. L. Richie, F. H. Mendoza, M. E. Enama, I. J. Gordon, L.-J. Chang, U. N. Sarwar, *et al.*, "Protection against malaria at 1 year and immune correlates following PfSPZ vaccination," *Nature medicine*, vol. 22, no. 6, p. 614, 2016.

[6] J. E. Epstein, K. M. Paolino, T. L. Richie, M. Sedegah, A. Singer, A. J. Ruben, S. Chakravarty, A. Stafford, R. C. Ruck, A. G. Eappen, *et al.*, "Protection against plasmodium falciparum malaria by PfSPZ vaccine," *JCI insight*, vol. 2, no. 1, 2017.

[7] M. S. Sissoko, S. A. Healy, A. Katile, F. Omaswa, I. Zaidi, E. E. Gabriel, B. Kamate, Y. Samake, M. A. Guindo, A. Dolo, *et al.*, "Safety and efficacy of pfspz vaccine against plasmodium falciparum via direct venous inoculation in healthy malaria-exposed adults in mali: a randomised, double-blind phase 1 trial," *The Lancet infectious diseases*, vol. 17, no. 5, pp. 498–509, 2017.

[8] K. E. Lyke, A. S. Ishizuka, A. A. Berry, S. Chakravarty, A. DeZure, M. E. Enama, E. R. James, P. F. Billingsley, A. Gunasekera, A. Manoj, *et al.*, "Attenuated pfspz vaccine induces strain-transcending t cells and durable protection against heterologous controlled human malaria infection," *Proceedings of the National Academy of Sciences*, vol. 114, no. 10, pp. 2711–2716, 2017.

[9] B. Mordmüller, G. Surat, H. Lagler, S. Chakravarty, A. S. Ishizuka, A. Lalremruata, M. Gmeiner, J. J. Campo, M. Esen, A. J. Ruben, *et al.*, "Sterile protection against human malaria by chemoattenuated pfspz vaccine," *Nature*, vol. 542, no. 7642, p. 445, 2017.

[10] T. Bousema and C. Drakeley, "Epidemiology and infectivity of plasmodium falciparum and plasmodium vivax gametocytes in relation to malaria control and elimination," *Clinical microbiology reviews*, vol. 24, no. 2, pp. 377–410, 2011.

[11] Sanaria Inc. (2014) SporoBot - Build a Robot. Fight Malaria. Save Lives! [Online]. Available: https://www.youtube.com/watch?v=VblazNXcHFg

[12] I. Lapowsky, "The next big thing you missed: This mosquito-dissecting, malaria-killing robot needs your help," Jun 2014. [Online]. Available: https://www.wired.com/2014/06/the-next-big-thing-you-missed-a-crowdfunded-mosquito-dissecting-malaria-killing-robot/

[13] C. Borchers, "Robot may help fight malaria," May 2014. [Online]. Available: https://www.bostonglobe.com/business/2014/05/07/mosquito-harvest/Qxto58qtpGHhRVfliT6aHI/story.html

[14] R. H. Taylor, A. Canezin, M. Schrum, lulian Iordachita, G. Chirikjian, M. Laskowski, S. Chakravarty, and S. Hoffman, "Mosquito salivary gland extraction device and methods of use," U.S. Patent Application US20 170 355 951A1, June 13, 2016.

[15] M. Schrum, A. Canezin, S. Chakravarty, M. Laskowski, S. Comert, Y. Sevimli, G. S. Chirikjian, S. L. Hoffman, and R. H. Taylor, "An efficient production process for extracting salivary glands from mosquitoes," *arXiv:1903.02532*, 2019.

Manuscript 475 submitted to 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Received March 19, 2019.

[16] M. Xu, S. Lu, Y. Xu, C. Kocabalkanli, B. K. Chirikjian, J. S. Chirikjian, J. Davis, J. S. Kim, S. Chakravarty, I. Iordachita, R. H. Taylor, and G. S. Chirikjian, "Mosquito staging apparatus for producing PfSPZ malaria vaccines," 2019, Submitted to CASE 2019.

[17] H. Wu, J. Mu, T. Da, M. Xu, R. H. Taylor, I. Iordachita, and G. S. Chirikjian, "Multi-mosquito object detection and 2D pose estimation for automation of PfSPZ malaria vaccine production," 2019, Submitted to CASE 2019.

[18] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan 1979.

[19] L. Feng, P. Wilkening, Y. Sevimli, M. Balicki, K. C. Olds, and R. H. Taylor, "Accuracy assessment and kinematic calibration of the robotic endoscopic microsurgical system," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug 2016, pp. 5091–5094.