KBBQ for KBBeginners

Joseph Jang, Jasper Lessiohadi, Raymond Ngo

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—Korean Barbeque is a delicious dish, but can be slightly daunting to people who have never experienced it before. To aid beginners through this, we implemented a robotics system that can help KBBQ beginners to properly cook meats. The system uses OpenCV and yoloV5 to incorporate computer vision, along with a robotic arm with precise inverse kinematics to scan various meats and automatically cook them, leading to food that is cooked well every time.**

***Index Terms*—Computer vision, design, inverse kinematics, robot**

I. Introduction

Korean Barbeque (KBBQ) is a type of meal where people grill their own meats on a grill in the middle of the table. However, those who are new to Korean BBQ could have difficulty properly cooking the different varieties of meats that are served. Each ingredient has a different thickness and required internal temperature, and beginners who have little experience may not know the appropriate amount of time to leave them on the grill. This could lead to burnt food or food poisoning if the dishes are cooked improperly. To avoid this situation, our group proposes a robotics system that can help KBBQ beginners to properly cook meats. The system uses a robotic arm with four degrees of freedom to handle putting the meat on the grill, taking it off the grill, and flipping the pieces to cook both sides. We also use computer vision to scan each ingredient as we put it on the grill to determine how long a certain dish has to be cooked, mainly based on thickness. When the meat needs to be flipped or taken off the grill, the system uses a robotic arm to do so. Meats are each placed in four sections of the grill to avoid the possibility of them overlapping. There is also an overhead camera feed of the grill on a touchscreen connected to the system. Users are able to tap on individual sections to signal to the arm that they would like meat in that section to be flipped, if they so choose. By using this product, users will eliminate the risk of overcooking or undercooking their foods, resulting in perfectly cooked meats every time. Some dishes we used for this project are pork belly (sam-gyup-sal), marinated ribs (LA Galbi), thin beef slices, and marinated beef (bulgogi).

The reason we chose these strategies in our solution is because we felt that fully automating the cooking process was the best way to ensure beginners have well-cooked food every time without having to worry about a new, possibly overwhelming experience. The advantage of primarily using thickness, rather than the type of meat, to determine cooking time is the fact that it is significantly easier to train our algorithm. If we wanted to consistently identify meat type, we would have to feed our algorithm much more data than we have, making this strategy unfeasible. Additionally, since the meats served in KBBQ are generally quite thin, the difference in internal temperature required to be fully cooked between beef and pork is less of a concern. Furthermore, we decided to use a robotic arm with a 'hand' instead of something like a spatula because we felt that it would be more precise. A spatula would also be significantly worse at picking up the raw ingredients, since they would likely be arranged as a pile on a plate, rather than a single piece on a completely flat surface. Overall, we felt that these choices were the best solution to creating a smooth experience for the user.

II. Use-Case Requirements

To consider this a successful project, we have several requirements. The first is that our computer vision algorithm must be able to correctly identify the type of meat being cooked within 5 seconds, 80% of the time. It must also be able to identify the thickness of said meat within 1/16th of an inch of error. Failing to do those two would mean that we are feeding incorrect data to the cooking time algorithm, leading to poorly cooked meat. Additionally, the robotic arm must be able to touch points on the grill within 1/16th of an inch of error. Having precise control of the arm is crucial to the function of the system, since we need to reliably pick up small, thin pieces of meat in a quick and efficient manner. Next, the touchscreen UI must correctly display the camera feed of the grill, and always register touches which signal that the user wants the meat in the chosen section to be flipped or removed from the grill. Finally, the algorithm to determine the cooking time for a given piece of meat must have a failure rate of less than 10%, where failure is either undercooking (internal temperature is too low) or burning the meat. The reason for this requirement is obvious: cooking the meat incorrectly is the whole goal of our project.

The entire system needs to last the length of an average dinner to be effective, which is to say it needs to last anywhere from 20 to 45 minutes. The system needs to simultaneously handle multiple pieces of meat at once to ensure enough food is cooked for an enjoyable dining experience.

III. Architecture and/or Principle of Operation

The image on the next page shows the basic setup of our operations. A metal heated grill exists. On the left side of the grill, the user can place meats they desire to cook on a plate designated as the raw meat plate. On the right side of the grill, cooked meats are placed on a dish designated as the cooked meats plate. In between the plates and in front of the grill, the robotic arm is placed so that it can reach any part of the grill with its claw. Not

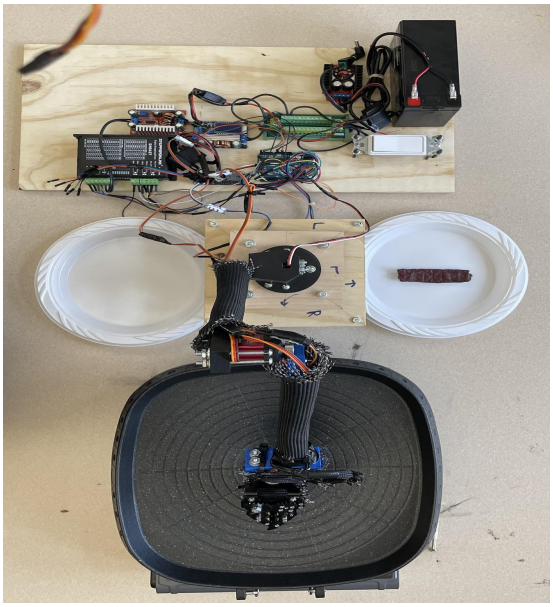


Figure 1: Robotic Arm system top down view. This displays the overall layout of our product

shown in the image above is the top camera, which is placed above the grill at a safe distance from the heat so that a bird's eye view of the grill is available. A side camera is placed at the same level of the raw meat plate to find the thickness of the meat. This operation setup allows us to simplify some of the implementations of the robot's kinematics and allow the cooking of KBBQ to be efficient and safe.

The KBBQ robotic system has have four main subsystems, which are the Robotic Arm, Computer Vision, UI, and Software Controller. In the diagram below, each box represents one or more subsystems. The red and green boxes, which are the 4 Degree-of-Freedom Robotic Arm and Power Distribution wiring, respectively, make up the entire Robotic Arm subsystem. The inverse kinematics algorithm within the Jetson AGX Xavier (orange box) is also part of the Robotic Arm Subsystem. The blue box that contains both cameras and the software subsystem (rounded box) within the Jetson AGX Xavier box that is labeled "CV Algorithm" represents the Computer Vision Subsystem, which we are naming the CV Algorithm subsystem. The Software controller box within the orange software block is the final subsystem, which is purely software.

As the key on the top right of the diagram, dotted lines within a colored box represent connections between the various software subsystems and algorithms. A solid line within a colored box represents physical electrical wirings within a subsystem. Blue lines are interconnections between different parts to the Jetson AGX Xavier, which is responsible for the software subsystems. Red Power lines represent physical electrical power lines from the power distribution block to the main subsystems. And finally, black solid lines that go past the dashed box, which represents our robotic system, are used to indicate inputs and outputs into and out of

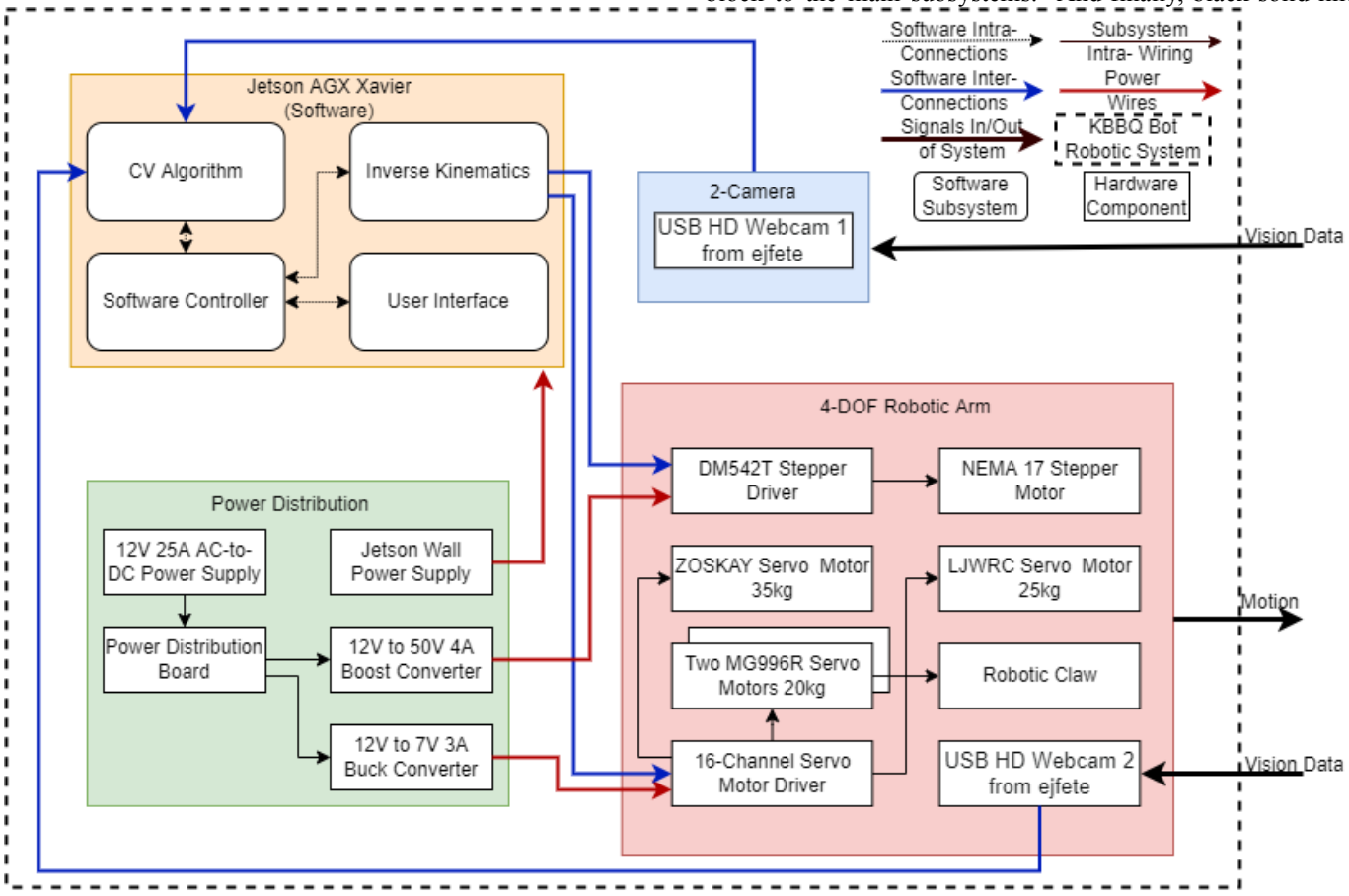


Figure 2: System architecture diagram.

our system, such as vision and motion. The changes to our System Architecture are shown above, with the replacement of the lead acid battery with a more reliable AC-to-DC power Supply, and we used two webcams for the CV.

I. System States

Figure 3 is a system state diagram for cooking a single piece of meat. Once an empty spot on the grill is found, the meat must be picked up from the left of the grill. The robotic arm holds the meat to the side camera to observe its thickness, which is necessary to compute the cooking time. Next, the meat is placed on the grill at the specific empty location to cook. After time is kept track of for the first side of the meat, and placed on the grill (about 3 to 5 minutes), it is picked up from the grill, flipped, and placed back on the grill. After enough time passes for the second side of the meat, the meat is once picked up again and placed on the dish that is on the right side of the robotic arm and grill. At this point, the system state diagram ends for that single piece of meat.

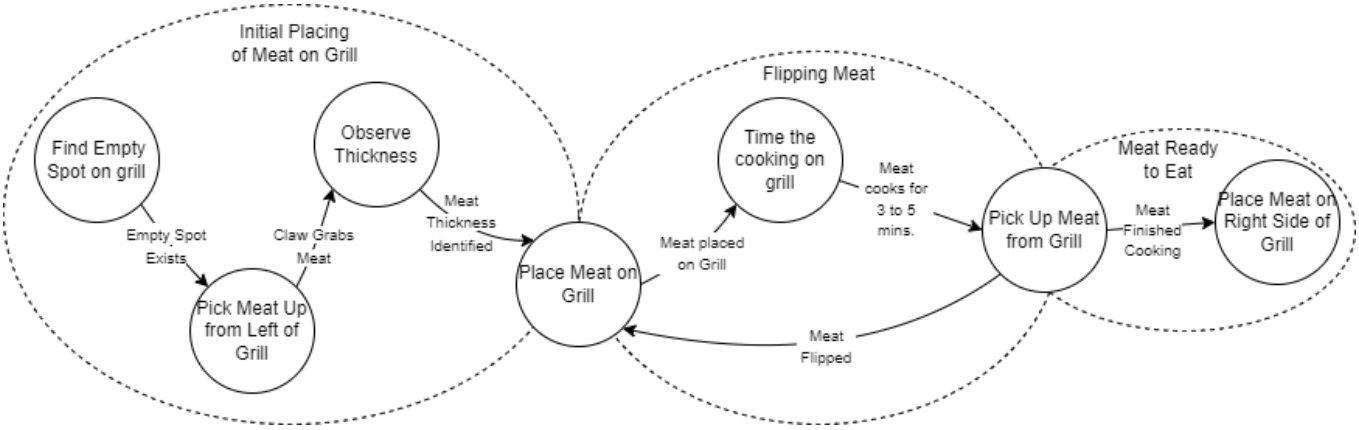


Figure 3: State diagram of flipping and placing single item

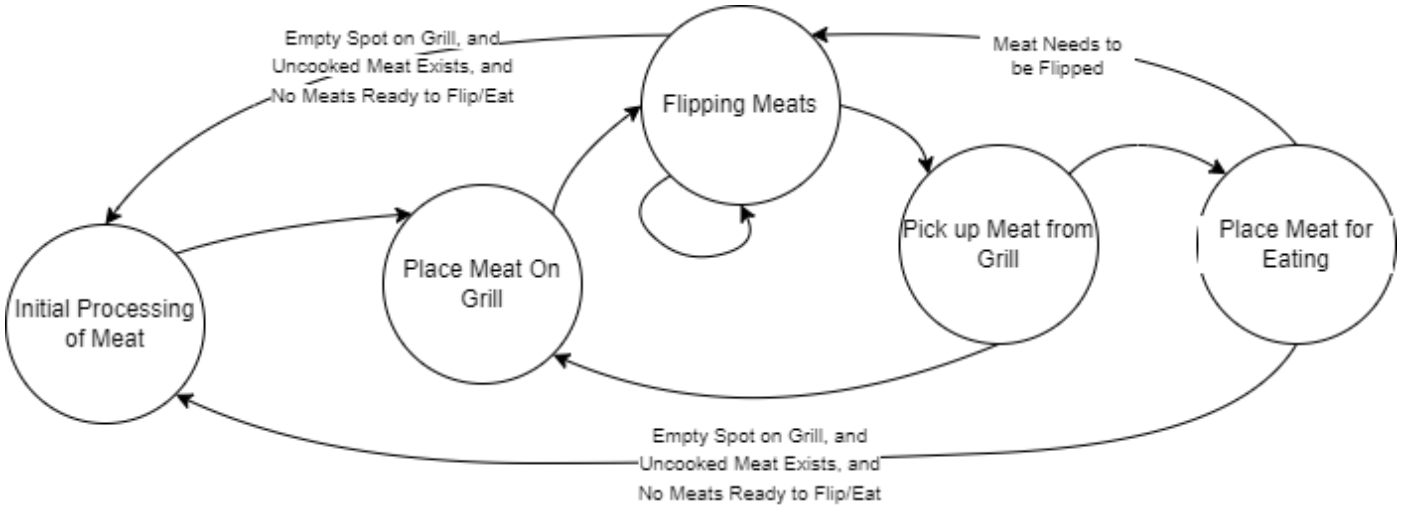


Figure 4. State diagram of flipping and placing multiple pieces of meat.

The second state diagram (figure. 4) handle multiple pieces of meat. It shows prioritization for flipping meats and placing them back on the grill or the cooked meat dish, away from the grill, over placing meats on the grill. Hence, the "Initial Placing of Meat on Grill" states will only occur if an empty spot on the grill exists and uncooked meat exists on the left dish and if no meats are ready to be flipped or pick up from the grill (all meats on the grill are in the "Time the cooking on grill" state in the individual state diagram). These state diagrams are used to help develop and verify that our robot is working as expected, first for one piece of meat on the grill, and then for multiple pieces of meat. Our software controller code very closely follows our system states plan. The only change is we added a default position state for the robotic arm.

IV. Design Requirements

The most important design requirement is being able to properly cook the meat the user shows the robotic arm system. The first main step in this process is proper identification of the types of meats placed on the grill. We plan on having at minimum 80% classification accuracy in our computer vision systems. Highly trained networks are able to classify between thousands of categories of objects with 99% accuracy, however, as we are providing our own dataset, as well as our lack of relative expertise compared to researchers, we lowered our threshold for accuracy to 80%. Furthermore, this recognition needs to be under 5 seconds, as studies have found out humans are impatient and 75% of people give up within 5 seconds of no loading on a web page, which is a proxy for

robotic arm onset movement time we have set.

Another important aspect of getting the proper cooking time besides classification accuracy is meat thickness precision. As some meats for Korean BBQ vary in thickness due to the nature of their cuts, it is important to obtain thickness measurements for certain cuts to obtain an accurate cooking time estimation for the food. As the thinnest cut of meat we plan on measuring is ⅛ of an inch thick, we expect the margin of error to be a maximum of 1/16 inch.

One important aspect of the entire solution, of course, is cooking. The cooking time is determined as a function of the type of meat as well as the thickness of the meat in certain scenarios. The function is not to rely on any input from the camera to ease project overcomplexity. The cooked meat should reach an internal temperature of at least 170 for beef and 145 for pork, and should not be burnt or undercooked. Our goal is to create an environment to lower the intimidation one faces when encountering Korean BBQ for the first time, so these requirements for internal temperature are necessary to decrease the risk of any potential issues such as food borne illnesses that decreases enjoyment factor.

The robotic arm must be able to reach a certain point with an accuracy of within 1/16 of an inch of a given selected point. It must have a maxim and minimum reach of at least 13” to 1” if the robot arm is placed right next to the grill. The electrical wires and components must be able to withstand grill temperatures of up to 400 F at any given moment or for a period of 1 minute. The robot must be able to complete a single action, such as picking up a piece of meat and placing it on the grill, within a 1 minute time frame. The robot must be able to move at least 12 cm/s. Lastly, the robot must be able to pick up pieces of meat that are at most 2 lbs. The battery power of the arm and the entire robotic system must last for at least 20 minutes to an hour.

V. Design Trade Studies

A. Processing Unit

The computer vision algorithm running many times a second to detect objects necessitates a strong mobile computing platform. The Raspberry Pi, without a dedicated GPU, is excluded due to its lack of strength compared to a product in the Jetson family of devices. However, the rapidity of the detection, the variety of our computer vision algorithms, and the strict time constraints required for computation mean the Jetson Nano might not contain the computing power necessary for our desired application. The Jetson Xavier, while potentially containing much more power than we need, provides more computing power and gives us breathing room should any operation require more computing power than we expect. Furthermore, the ability to loan devices instead of acquiring it with our funds frees our budget as part of our risk mitigation plan in the case where one of our components fails. As a result, price was not a factor in our comparison.

	Jetson Xavier	Jetson Nano	Raspberry Pi 4
CPU	8 Core Nvidia Carmel Cores (1.37 Ghz)	Quad Core A57 (1.4 Ghz)	Quad Core A53 (1.5Ghz)
GPU	512 core Volta GPU with 64 Tensor Units	128 core Maxwell GPU	None
Memory	16GB (136 GB/s)	4GB(26 GB/s)	2GB
Connectivity	3x USB 3 (2 of them USB-C)	4x USB 3	2x USB 2 2x USB 3

The result of this selection we made was that the Xavier was a better option despite the downside in fewer USB-A ports. We were able to overcome this weakness by using USB-C hubs to connect to the peripherals. Xavier had pin connectivity to make controlling the robotic arm easier. On many CPU bound functions, the Xavier performed similarly to our laptops, which made predicting performance on the Xavier easy in times where access to the Xavier was limited. The Xavier also made processing the YOLOv5 algorithm on pytorch much faster due to the speedup from its CUDA cores.

B. Computer Vision Algorithm

For the edge detection algorithm, the Canny edge detector[1] in the OpenCV[2] library showed itself best for our purposes. It accurately detects edges and has manipulable parameters. In addition, the use of built in functions allows us to avoid having to spend time creating edge detection algorithms from scratch using gradients.

For the object detection algorithm, the simpleblobdetector function presented similar benefits for the Canny edge detection in its simplicity compared to its competitors. One alternative we considered was using the classification algorithm to also detect the presence of objects, however that was ruled out due to the need to train for such a scenario. After brief preliminary testing, it would seem some form of image manipulation would need to be made to accommodate such a function. We plan to isolate red elements of the image to improve accuracy in object presence detection.

We needed a way to classify our images, and we had a selection of choices. Selecting a neural network was not among our top choices due to the size of the data set needed. However, many other methods we looked at that involved feature extraction requires fine tuning of parameters, and that is extremely difficult when classifying images that have extremely similar characteristics, such as, for instance, color and shape. As a result, we came to the conclusion that a neural network that learns the features could, despite the tradeoff in dataset size, ease the process in feature selection. Initially, in

our detailed design, we limited ourselves to using a classification algorithm to classify images. We decided against that, and decided instead to use an object recognition algorithm called YOLOv5[3] instead. The reason is that object recognition projects and libraries are much more easily accessible. The other reason is that the had a few cases (meat) where we want to properly classify, and a whole bunch of instances (empty plate, arm over plate, claw over plate, etc etc) that we would classify as 'other'. Having object recognition prevents networks from simply classifying as much stuff as possible as the 'other' category.

C. *Software Controller Operations*

For the cooking mechanism, we settled on a function that takes in the type of meat and its thickness and outputs out cooking time, which is fed to the robotic arm queue. Flipping time occurs halfway into the output cooking time. One key assumption made is the grill maintains temperatures around 500F. This is a key assumption made because KBBQ grills usually exist around that temperature range. Furthermore, temperature sensors from our research of different options max out their operating temperatures at or below 500F, which makes a grill that exceeds that temperature extremely dangerous for the sensor, which unfortunately rules out most sensors that fit within our desired budget. A computer vision algorithm to detect burning and meat doneness based on color was discussed, however such a solution would take time to train or tune. Most importantly, external meat color does not correlate with meat doneness. Meat can appear burnt but still not have a safe internal temperature.

This suspicion was confirmed during testing. If we used a computer vision implementation, the meat would've been marked as "done" right after being flipped, because the cooked side of the meat would've looked done, even though the other side of the meat was still clearly raw.

VI. System Implementation

Our project is arranged such that the robotic arm is on one side of the grill, a plate with raw meat is on its left, and a plate for cooked meats is on its right side. A side view camera exists next to the raw meat so that we can use blob detection to see how thick the meat is when the robotic arm picks it up. A third camera was placed above the grill for use to be funneled to the user interface. While it worked during a last minute test, we removed it to make repairing and moving the overall system easier. All of the processing involved is done using a Jetson Xavier, which is more than enough processing power for our needs.

A. *Computer Vision Algorithm*

The computer vision algorithms we used involve an edge detection algorithm, an blob detection algorithm, and an object classification algorithm. The edge detection algorithm uses a bounding rectangle around the dimensions of the meat to determine the pixel width, and by extension, the thickness of the object. Because the robot holds the meat up for the detection, the variance of the meat's distance away from the camera is bounded by the width of the meat where the robotic arm can grab.

The blob detection algorithm comes from openCV. Before the activation of the blob detection algorithm, the input image's color space is converted to HSV, then filtered to only highlight red portions of the image. After dilation and floodfill, any object with a red outline (in this case white), would be white, and every other object in the image would be black. This makes the job easier for the object detection algorithm by specifically highlighting red objects, which in this case would be meat.

The classification algorithm would try to find out which type of meat was placed in front of the camera. The classification is composed of a neural network and the dataset is created by ourselves. Our original plan was for images to be flattened and downscaled before being placed into the neural network. However, throughout the course of the project, we realized creating a network by ourselves instead of using a pre-existing algorithm was much more work than we expected. As a result, we switch to using a Pytorch algorithm called YOLOv5. The great advantage of yolov5 specifically was its speed and its extremely active support community. To get the proper classification label, we run YOLOv5 on the image and take the label with the highest confidence. One of three classifications would emerge as a result, as shown in the figure.

The YOLOv5 weights were trained ourselves and involved creating a database of both images found online and images we captured ourselves. Due to the difficulty in finding niche categories of images, this meant collecting data had to be found manually; there was no pre-made training set of images of raw Korean Barbecue meat. The dataset was rather small, comprising 40 images augmented to 120 images for the training set and a validation set of 20 images.

On the side camera, the object detection algorithm loops to detect if the user has placed meat in front of the camera. On detection of the meat, the classification algorithm takes an image from the camera and classifies the meat into several categories highlighted in Fig _. On classification, if the meat is classified as short rib or any type of meat where thickness is a necessary factor in cooking, the robotic arm lifts the meat vertically in front of the camera for the edge detection algorithm to detect the thickness of the meat.

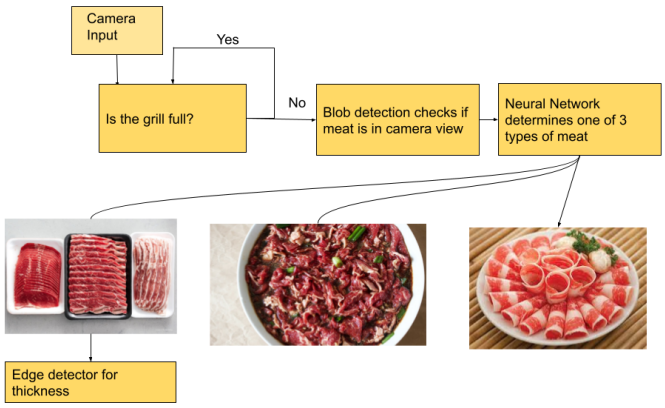


Fig. 5. Computer Vision algorithm flowchart. Note only one

classification of meat requires checking for thickness.

B. Robotic Arm

The robotic arm has 4 Degrees of Freedom. Therefore, it has 3 links and 4 joints. The first DOF is the base joint. This is the yaw movement, and is capable of freely rotating (360). The robot has two elbow joints, which accounts for the Pitch movement of the robotic arm. Each joint is capable of 270 of movement. Finally, the wrist of the robot, or the joint where the last link 3 is connected to the robotic claw, is the Roll (4th DOF), and is able to rotate 360. Link 1, 2, and 3 stand at a height of 3, 11, and 9 inches respectively. The claw is 4 in. long and can open 180 degrees. The CAD models of each link have been created in Solidworks in millimeter format. These links can now be 3D printed in Techspark using a Dremel 3D40 or F170 Printers. We used an infill of 20% and a precision of 0.01mm to print the 3 parts.

Figure 6: CAD model of robot arm.

The assembly of the physical robot was not difficult. The base was created using pieces of scrap wood from Techspark. We created a sort of box where the lazy susan bearings, stepper motor, and link 1 is attached to. However, the bearing was actually inhibiting the movement of the stepper motor, so it was removed, and instead the link 1 rests secured directly to the shaft of the stepper motor. Then, the 35kg-cm servo motor is bolted onto the four holes provided in link 1 using four 8/32 by 2 in. bolts and nuts. These same bolts are used to secure the 25kg-cm servo motor to link 2, and the MG996R 10kg-cm servo motor to link 3. A 25-teeth aluminum servo horn metal steering arm connects the ends with the smaller grooves of link 2 and 3 to the 2 strong servo motors. A 25-teeth disc horn then connects the MG996R servo on link 3 to the claw, which is powered by a MG996R servo motor.

For the electrical wiring of our robotic arm, a 12V 7Ah Lead acid battery was used to power our system. But due the limited current draw of the battery and it quickly depleting power in a few hours, it was replaced with an AC-to-DC Power supply, capable of supplying 12V and a maximum of 25A. An electrical power terminal block that can handle 30 Amps and 18 AWG wires is used to power various components of the system. A wall-plug in power supply powers the Jetson AGX Xavier. One boost converter powers the DM542T stepper driver and motor (needs around 20-50V). A buck converter is used to power the I2C servo motor controller, which requires a voltage of around 7 V and supports 16 servo motors (we only use four of these triple pins). All signal wires from the servo motor controller and the stepper driver are connected to the 40 pins of the Jetson AGX Xavier. The two cameras are connected to this computer. ATC fuses and fuse holders are used to protect electrical components. A heat sleeve, which is usually used to protect wires in car engines that could reach temperatures as high as 1000F, are used to cover all servo motors and wires so that no melting or electrical damage can occur. An Arduino Uno controls the stepper motor. An updated electrical diagram is provided at the end of this document.

The robotic arm stepper and servo motors can all be controlled using PWM and I2C. For the stepper motor, we used an Arduino Uno to accurately control the base of the robot, but the servo motors were controlled using the I2C servo motor controller we bought. As for the camera that was mounted on the robot, it was mainly used for depth perception when the robotic claw must pick, place, or flip a piece of meat.

Inverse Kinematics is when given a point in 3D space and the lengths of each link, the inverse kinematics algorithm can calculate the appropriate angle each joint must be at so that the robotic arm can properly move its end effector to that point in space. With the dimensions of each link and the claw, the robotic arm is able to have a maximum reach of at most 24 in. and a minimum reach of at least 4 in around itself (think of a circle with a radius of 4 in. and 24 in.). This is plenty of range, because the KBBQ grill is a circle with a diameter of 13 in. If we place the grill and plates 4 inches in front of the robotic carm, the robot should have no problems with the range of motion possible. To solve this inverse kinematic problem, there are many available resources. We have looked in ROS, Matlab, and Python libraries that have inverse kinematics solvers. We tried using the Robotics Systems Toolbox in Matlab, but we were not getting the desired angles so easily and consistently. We tried changing to ROs, with little luck, and even tried some Python libraries, such as IK Fast, IKPy, and tinyik. In the end, we decided to implement the IK algorithm ourselves. The robotic arm has four joints.

One is the base, and the other is the wrist, which can be separately controlled. That leaves us with two revolute joints. I then realized that oru inverse kinematic algorithm can simply be an RR robotic arm algorithm, which is a solved and easy problem. Hence, after some programming, the robotic arm could accurately move to the proper position with the appropriate angles. To help constrain the environment the robotic arm would work in, we decided on the (x,y,z) origin point of the robot to be the second joint, or the first joint where the first servo motor is, which is also directly above the first joint, which is the stepper motor. This origin point is labeled in the pictures below and next to this text. This origin point allowed the environment to be split into 2 coordinate planes (xy and yz). If the meat pieces have a maximum dimension of 2 in by 5 in, and the grill height is 13.5 cm, the 2 coordinate planes would be the xy plane (horizontal, parallel to floor), and a yz plane (vertical, parallel to wall). The plates (diameter of 9in) are at 0 and 180 degrees (directly left and right) of the base of the robotic arm.

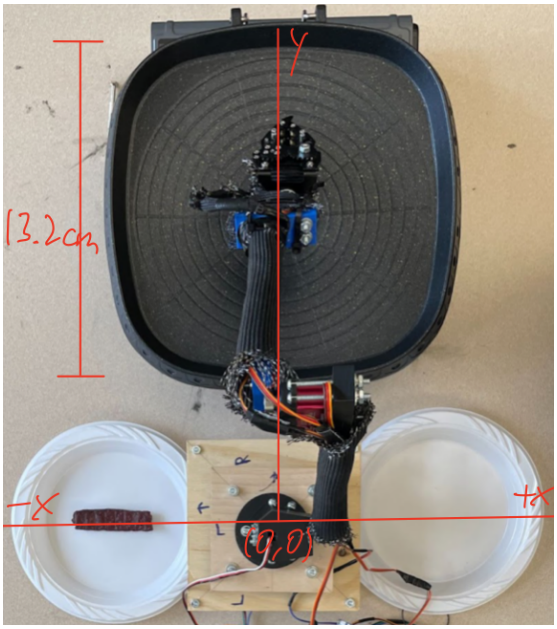


Figure 7: Overhead view of grill, arm and plates

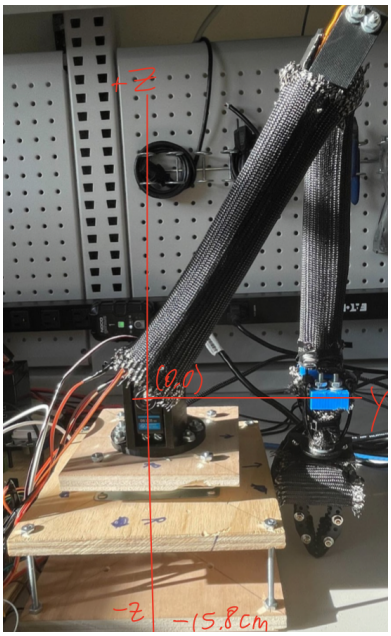


Figure 8: Side view of the robot arm

C. *User Interface*

Our UI involves a touch screen which displays an overhead camera feed of the grill, along with some other information. The camera feed is split into four sections, and the user can tap on any section to manually signal to the robot to either flip or remove the meat on that section, depending on which action needs to be done. Information regarding the remaining amount of cooking time is also displayed so that the user knows how long they have to wait until their food is ready.

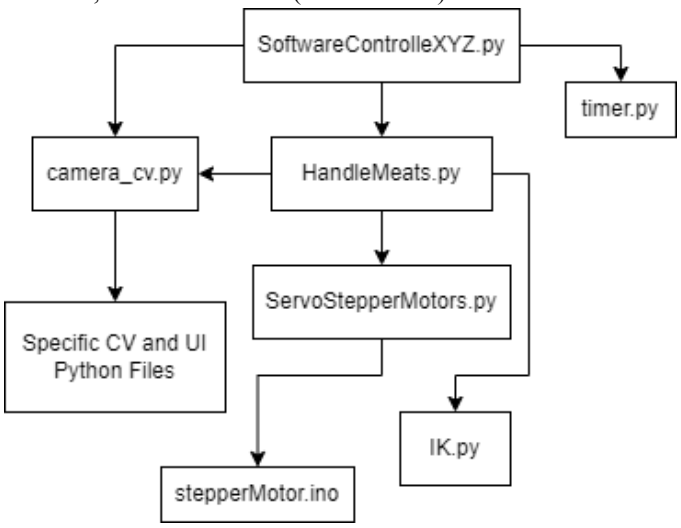
We used PySimpleGUI for most of the components of the interface, along with openCV for the camera feed. We did have some difficulty integrating this part of our system, since the camera feed here along with the computer vision subsystem had to be run on the same thread. Because of this, we had to take out the camera feed on the UI subsystem in order to keep the essential information that we wanted to deliver to the user while still enabling the computer vision to do its job.

s

Figure 9: Layout of UI design. The mentioned camera feed is on the right, while information on the newest meat to be put on the grill is on the left.

D. *Software Controller*

This subsystem is the main subsystem to integrate all the other subsystems - CV, Robotic Arm, and UI. It takes in information from the CV Algorithm, and displays CV data to the UI. It needs to keep track of multiple meats on the grill and display their cooking times. It needs to calculate information for each meat and make decisions for each meat. The software controller must also be able to make decisions, such as providing a desired action and position to the inverse kinematics algorithm. The controller prioritizes flipping and getting meats off the grill at the appropriate time over putting meats on the grill. It uses two queue implementations. One queue holds events related to robotic arm actions meat on the grill, while the other holds events related to robotic arm actions related to meat about to be added to the grill. The queues are separate to ensure actions relating to meat already on the grill take priority over meat yet to be cooked. The controller understands that the dish of uncooked meats that need to be grilled is at the left of the robotic arm and grill, while cooked meats are placed on a dish at the right of the robotic arm and grill. In essence, the software controller is truly the brains of our entire robotic subsystem. It is where all the other subsystems integrate with one another and must also work with one another. Below is an image of the software hierarchy of our robotic system. The SoftwareControllerXYZ.py was our main controller, and arrows indicate what module (start of arrow) gave instruction, or called, to which module (end of arrow).



During implementation, there were challenges in integrating the User Interface and the other elements of the system, due to the User Interface's need to see everything at all times. A workaround we used was attempting to have the user interface in a separate thread, but the issue was the updating of the meat thickness display relies on information from the thread with the CV and robotic arm functions working, in essence bottlenecking performance. As a result, the user interface only displays camera information. Due to this reduced functionality, we decided not to showcase it for demo day, preferring instead to dedicate the limited amount of table space for more crucial parts of the arm.

VII. Test, Verification and Validation

A. *Results for computer vision subsystem*

The computer vision system had several tests for its multiple components. To test the classification strength of the object, as well as its ability to predict in real time, meat would be placed in front of the camera, and the network's prediction class and its prediction time would be outputted. Using this method of testing, a desired accuracy of 80% and a desired processing time of under 5 seconds would need to be obtained.

Blob detection was meant to be slotted in at twice a second. Instead, a Python performance test pegged it at .025 seconds, which made it much faster than even our goal. This timing was consistent over many runs, therefore, there is no results chart as with the other tests.

To test the accuracy of the measuring system, a section of a table was marked, a person (or a hanging clip) held the meat vertically in front of the camera. The distance the meat is from the camera does not matter so long as the clip or hand remain the same distance from the camera between tests. The predicted measurement result is compared against the actual measured thickness.

For the neural network, our confusion matrix yielded an accuracy of 77%, which is slightly lower than the benchmark we set for ourselves. The reason seems to be that there are instances where the different types of meat appear really similar and may have tricked the network, which could possibly be because of the small dataset. Due to the difficulty in obtaining images, we only had a dataset of 40 images augmented to a train set of 120. Speed greatly surpassed our goal of 5 seconds, recording a .05 second time. Despite this, the blob detector was still twice as fast as the network.

For the edge detector, the thickness bounds was 1/10th of an inch. As we continued our project, we discovered 1/16th of an inch margin of error was possibly too wide considering lighting conditions and a limited resolution of the camera. We tested by measuring the width (not thickness) of the piece of meat in front of the camera, held 4 inches away. We measured width because we did not know how big the error bound was before testing, and in the cases where the bound was 1/8th of an inch, ⅛-⅛ is 0.

		Actual				
		Slab	Blob	Round	None	Pred Total
	Slab	13		2		15
Predicted	Blob	2	6			8
	Round	2	1	9		12
	Miss	1		1	2	4
	Total Actuals	18	7	12	2	

Fig. _. Compilation of test results for the object detection

Trial Num	Actual	Recorded	Diff	Average	0.10625
1	2.52	2.47	0.05	Std	0.068909
2	3.11	2.99	0.12	Var	0.004748
3	2.41	2.43	0.02		
4	3.22	3.28	0.06		
5	3.34	3.41	0.07		
6	2.84	2.72	0.12		
7	2.91	3.16	0.25		
8	2.97	2.81	0.16		

Fig. 10. Compilation of test results for thickness recognition

B. *Results for Cooking Time algorithm*

In this test, the cooking time function was called with a specific meat type and thickness as the manual input, as opposed to the final system where the computer vision algorithm provides the function inputs. We then cook the meat on the grill in accordance with the time output from the function, flipping at the exact halfway point. The test was conducted inside a group member's kitchen. While the original test had a meat thermometer to test the doneness of the meat, we did not purchase the item, preferring to spend the budget on other items more relevant to the robotic arm. Instead, we used the eye test to see if meat was done. This test was actually done a lot less and a lot more sparingly than we would have liked, mainly because we recognized at the end of the semester that grilling meats during the demo causes more problems than we would have liked. This suspicion of ours was backed up when the servo motor began overheating after testing the arm on the grill, and we suspect the residual heat from the gas burner may have made the servo motor's overheating problems worse. Overall, 0% of the meat was undercooked (defined as pink in the meat), and 15% were slightly charred but not burnt. No marinade was used, it was just the meat on a pan.

Meat Type	Outside Condition	Inside Cooked
Slab (1/8 inch thick)	Cooked	Cooked
Slab (1/8 inch thick)	Cooked	Cooked
Slab (1/8 inch thick)	Cooked	Cooked
Slab (1/8 inch thick)	Cooked	Cooked
Blob	Cooked	Cooked
Blob	Cooked	Cooked
Round	Cooked	Cooked
Round	Charred	Cooked

Fig. 11. Data for the cooked meat results

C. *Results for Robotic Arm and Software Controller*

The first test we conducted using the robotic arm was motor tuning. Using a protractor and a measuring tape, specific (x,y,z) points in space were decided upon, and once that point was put through the IK algorithm, the specific angles were measured. I mainly used 0, 45, 90, 135, and 180 degrees input into the motors, and measured whether these angle errors were within 5 degrees of accuracy using a protractor. This testing phase was probably the longest, taking several days to properly tune each motor. This was also part of the design and implementation phase of the inverse kinematics algorithm. After each servo motor was tuned, the stepper motor was tuned.

Next, after all motors could accurately and precisely move to certain angles and positions, the three behaviors of the robot were tested. For the pick up, flipping, and drop tests, they were conducted primarily on pieces of beef jerky instead of raw meat. The robot had to pick up raw meats and place them on the grill, flip those meats, and also pick the meats from the grill and place them on the plate in the correct order specified in our system states diagram. Out of 10 tries, the robot could properly pick up meat from the raw meat plate and place it on the grill 8 out of 10 times. Flipping the meat was more difficult, so the robot could flip the meat on a grill successfully 6 out of 10 times. But the robot could properly pick up meat from the done meat plate and place it on the grill 9 out of 10 times.

Each robot task was also timed. The raw meat task could be handled in about 30 seconds, flipping a meat was handled in about 25 seconds, and the done meat task was handled in about 30 seconds. All these tasks were initially under 20 seconds to complete, but we discovered making tasks that quickly lead to more mechanical motor errors, current draw issues, and position inaccuracies.

VIII. Project Management

A. *Schedule*

The end of the final report shows the actual schedule. We had to push back integration well into finals week, and even into the day before the demo. All of the individual component testing had been done well before integration. The reason for the delay compared to the original goal was because inverse kinematics took a lot more time due to difficulties involving potential libraries and required us to switch between entirely different programming languages.

B. *Team Member Responsibilities*

During the design phase of the project, the workload balance was designated as Joseph doing the robotics and the inverse kinematics, Raymond implementing the computer vision, and Jasper implementing the UI and cooking time functionality. All members involved would work on the integration and the controller together. In the course of the project, several issues popped up that necessitated changes in this workload balance. Unfortunately, Jasper was ill with a COVID-related illness during the integration phase, which removed him from most of the integration operations. The process of integration of systems came down to Raymond and Joseph, with Joseph primarily responsible for integrating the CV into the robotic arm functionality and Raymond coordinating all other aspects of the integration, which included making potential modifications (such as making libraries available offline) needed to make the system suitable for the demo area.

C. *Bill of Materials and Budget*

Please refer to the bill of materials table below for the materials. Note that during the process of development, we retired the lead acid battery from service for a DC power supply that plugs into a wall outlet, primarily because of current draw issues from the battery and a recognition that there would be power in the demo space.

D. *Risk Management*

For risk management, we mainly had multiple parts we borrowed to use in case of a breakdown. The only risk

management for our software was just having a GitHub repository of our work.

For the power supply, our initial risk was potential failure using those components. Current draw from the lead acid battery was lower than we would have liked, and as a result we had to switch to a different DC power supply. The one that was bought was the AC-to-DC power supply mentioned before, and it worked wonders.

For the risks defined in the risk mitigation of the detailed design writeup, several of them popped up in the operation and construction of the robotic arm. The robotic arm once chipped while the servo motors were not controlled properly and hit the arm against the desk. We had to use a soldering iron and some leftover PLA plastic to essentially reform the shipped parts so that the servo motor could be secured properly back onto the plastic link of the robotic arm.

It would be remiss not to mention the issues with reliability of several parts. Servo motors, stepper motors, drivers, and Arduinos all broke at some point or another during the course of the project. The broken Arduino could not manage global variables correctly, which was a bug that took about 9 hours to figure out. And on the day of the demo, a broken motor on the day of the demo itself caused much headache. We diagnosed the problem as excessive heat, which damaged the motor and the stepper driver. Due to their integral part of the design of the robotic arm, there were no substitutes except replacement of the part with an identical but functioning part. Perhaps with a higher budget a chance to obtain more reliable parts could have happened.

For the cooking time risks, the detailed design writeup mentions the risk involved in the potential inaccuracy of the meat cooking algorithm. Fortunately, that has not been the case. Had the function not worked, we could have tweaked the function to be better at preparing the cooking time.

Despite us wanting the focus of this project to not be mechanical, but more electrical and software, mechanical risks took the longest to debug, mitigate, and fix. However, our risk management plan prevailed in the end, and we were able to fix most if not all issues we encountered.

IX. Ethical Issues

One of the primary potential ethical issues we encountered is its potential unfriendliness to people with children. Our concern was that children would get their arms stuck in the robotic arm and get dragged to the grill. Our intention was never to cause any bodily harm or injury to the users, to help create a more fun experience for those involved. Our solution involved tweaking some aspects of the design. For instance, the computer vision algorithm was trained in such a way that it avoids recognizing human arms as meat. The idea behind this is to be sure that the arm grabbing a human in an accident is not the cause of the software malfunctioning. Our next challenge involved the robotic arm assembly itself. Children are nosy creatures and have a tendency to grab things even if they are dangerous. Potential pain points include the hot arm claw and the exposed robotic arm body. We solved this by covering most of the arm in a sleeve that removes most of the potential danger areas for a child to touch. We also designed the arm in such a way to operate over a hot surface of a grill without malfunctioning, and the arm claw should not reach dangerous temperatures.

Unfortunately, there are not a lot of ways to relieve a situation where a claw accidentally grabs someone's arms. The claws have a necessarily strong grip to counteract the softness and slipperiness of the meat, and the forceful opening of the claws breaks the servo motors. Furthermore, there is no override while the arm is in the middle of an action. However, we consider this to be an inevitable part of the pitfalls of cooking in this fashion. In the same way you can't prevent children from doing dangerous things such as touching a barbeque grill during a cookout, we have to accept there are limits to how much we can child-proof our devices without compromising on functionality. That is the tradeoff we had to make regarding ethics. Therefore, we advise placing meat on the plate using tongs, and to avoid having children near the robotic arm.

X. Related Work

As mentioned in the design review, the main point of comparison for a similar project is Hot Pot Bot. Both projects involved making the process of cooking a dish meant to be eaten communally from an East Asian cuisine using the use of computer vision to help identify food and the use of a mechanical apparatus to aid in cooking.. This similarity persists even into the completion of the project.

XI. Summary

Overall, our project aims to assist those who would like to try delicious KBBQ, but may get overwhelmed by such a new and different experience. To do this, we have designed a system to automate the cooking process so that users can enjoy the company of those they are eating with, rather than stress about how long to leave ingredients on the grill. A robot arm handles flipping the meat, along with moving it on and off the grill. Using CV, we were able to reliably determine the correct amount of time to cook the ingredients presented. In the case that the time is calculated incorrectly, though, users had the option to use the touchscreen UI with a camera feed of the grill to manually tell the system to flip or remove a specific piece of meat. In this way, we gave users an efficient and streamlined way to enjoy KBBQ.

In the process of implementing our design, though, we ran into several challenges. Having IK for the robot arm and blob detection for our CV that are precise enough to fulfill our design requirements was not easy; however, in the future with more testing, we are confident that we will be able to find ways to make them work at a satisfactory level. In terms of the cooking time algorithm, we will need to perform a lot of time consuming testing to make sure that we do not serve undercooked or burnt meat. Despite these challenges, though, we know we can produce an effective system that will help anyone who would like a more stress-free experience with KBBQ.

For the robotic arm itself, the biggest issue was mechanical

and then electrical. Because not wanting the focus of this project to be a mechanical design of a robotic arm, the implementation of the physical robotic arm was done rather quickly. However, multiple issues that took hours and days to debug appeared because of motor issues, especially because of the overheating of the stepper motor. In the future, I would definitely want to take a much closer look into the mechanical constraints of a robotic arm and the realities of implementing an arm, which would help with the ideal mechanical version of a robotic arm I understood and learned about heading into this project.

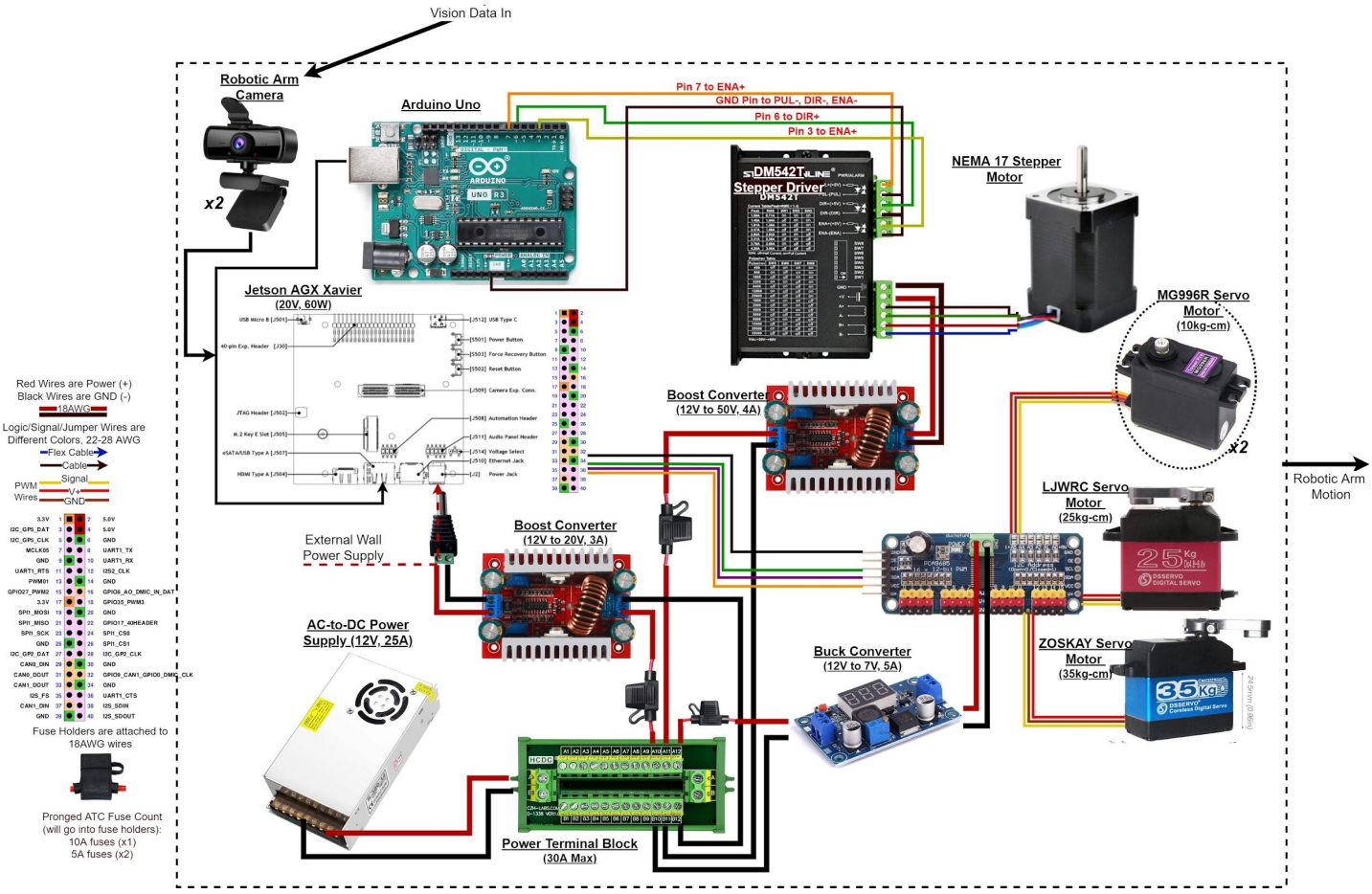
Over the course of the semester, we learned that allocating more time to integrating all of our subsystems and communication are necessary because it was more difficult and time consuming than we initially expected. We also learned the importance of communication between team members and establishing a shared goal for the duration of the project.

Glossary of Acronyms

CV - Computer Vision
DOF - Degrees of Freedom
IK - Inverse Kinematics
KBBQ - Korean Barbeque
UI - User Interface

References

[1] J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.
[2] *OpenCV.* url: https://opencv.org/.
[3] *YOLOv5*. url: https://github.com/ultralytics/yolov5



Electrical System Design

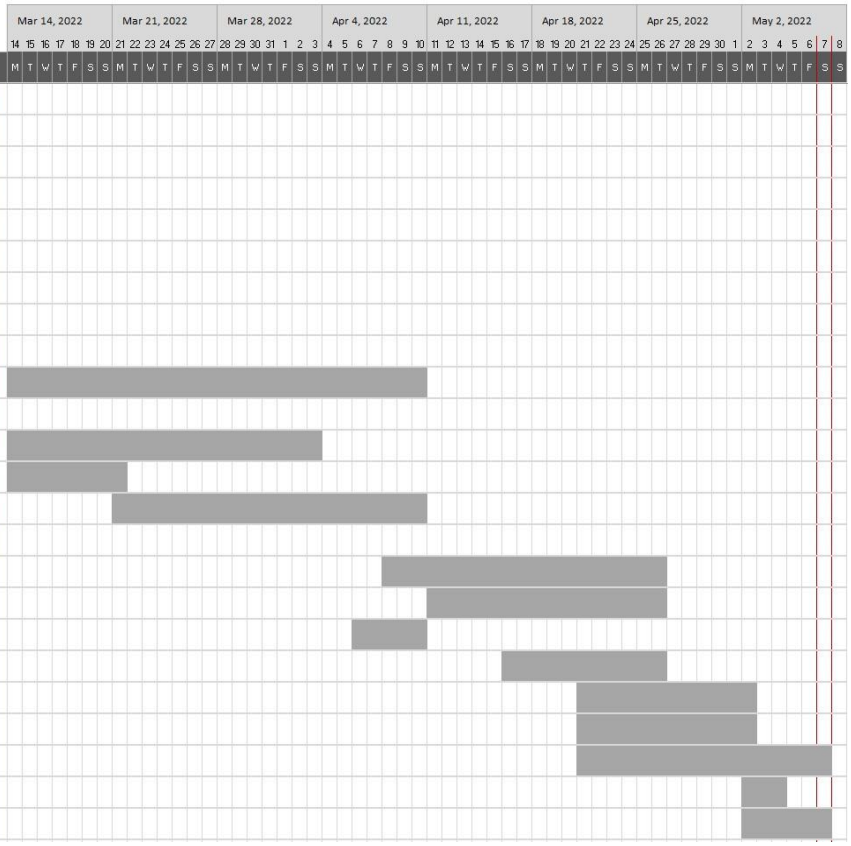
KBBQ 4 KBBeginers

Team B5

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Project Start: Mon, 2/14/2022
Display Week: 5

TASK	ASSIGNED TO	PROGRESS	START	END
Phase 1 Design				
Task 1	Confirm System Architecture	100%	2/14/22	2/19/22
Task 2	Research and Design CV Algorithm	100%	2/14/22	2/24/22
Task 3	Design Software Controller/UI	100%	2/14/22	2/24/22
Task 4	Design and CAD of Robotic Arm/Controls	100%	2/18/22	2/25/22
Task 5	Create Subsystems Integration Plan	100%	2/25/22	3/2/22
Task 6	Design Review Presentation	100%	2/15/22	2/20/22
Task 7	Design Review Report	100%	2/15/22	3/2/22
Phase 2 Development				
Task 1	Develop CV Algorithm	100%	3/14/22	4/10/22
Task 2	Slack Time (Spring Break - Design, Develop, etc)	100%	3/2/22	3/13/22
Task 3	Develop Software Controller/UI	100%	3/14/22	4/3/22
Task 4	3D Print/Assemble Robotic Arm	100%	3/14/22	3/21/22
Task 5	Program Robotic Arm	100%	3/21/22	4/10/22
Phase 3 Integration and System Testing				
Task 1	Integrate Software Controller and CV Algorithm	100%	4/8/22	4/26/22
Task 2	Integrate Software Controller and Robotic Arm	100%	4/11/22	4/26/22
Task 3	Slack Time (Integrate, Develop, etc.)	100%	4/6/22	4/10/22
Task 4	Integrate All Subsystems	100%	4/16/22	4/26/22
Task 5	Testing CV on Actual Grill and Meats	100%	4/21/22	5/2/22
Task 6	Test Robotic Arm on Actual Grill and Meats	100%	4/21/22	5/2/22
Task 7	Complete Final Presentation	100%	4/21/22	5/7/22
Task 8	Complete Video/Poster	100%	5/2/22	5/4/22
Task 9	Complete Final Document	100%	5/2/22	5/7/22



SCHEDULE

Bill of materials

Part Name	Quantity	Status	Cost
Stepper Driver DM542T	1	Borrowed	-
NEMA 17 Stepper Motor	1	Borrowed	-
Lazy Susan Bearing	1	Owned	-
Wood, Nuts, Bolts	Several	Borrowed	-
Stepper Motor Mounting Connector	1	Owned	-
Servo Motor 35kg	1	Buy	$32.99
Servo Motor 25kg	1	Buy	$18.99
25T Disc Horns	1 (pack of 5)	Buy	$6.99
Robotic Claw	1	Buy	$27.99
KBBQ Grill	1	Buy	$29.98
Jetson AGX Xavier	1	Borrowed	-
Camera 1 USB HD Webcam from ejfete	1	Borrowed	-
Camera 2 USB HD Webcam from ejfete	1	Bought	24.99
Power Terminal Block	1	Owned	-
12V 25A AC-to-DC Power Supply	1	Owned	-
12V 7Ah Lead Acid Battery	1	Buy	$24.14
Boost Converter	2	Buy	$14.29
Buck Converter	1	Buy	$15.88
Heat Sleeve	1	Buy	$10.99
3D Prints	***Cost: ~$50-$100***	***Total***	***~$238.21***