Building an Electrician's Robotic Assistant: Optimizing Vision for Real-time Feedback

Kaleb Gezahegn

Advisor: Brian Scassellati

Abstract

Professor Scassellati's social robotics lab has been working on a project that is attempting to build a robot to assist and protect an Electrical technician. The physical environment for an electrical technician's workspace is simplified for the sake of the project, yet it is still very difficult to represent virtually. Not only are there multiple components within the junction box and tools that need to be kept track of, but the human agent also needs to be tracked while moving and interacting with these different components. The extensive nature of the environment and the distribution of its components require that the robotic system employ machine learning models on different camera angles to observe and monitor the surroundings from various perspectives. This requires a lot of processing which is problematic when designing a system that requires accuracy and real-time operation. This research paper focuses on two different approaches to addressing the problem of this processing bottleneck. The first part attempts to use image stitching to combine different camera perspectives into one feed to remove the need for running the same body and face-tracking models across two different feeds. The second attempts to establish a global coordinate system using aruco codes for the two cameras. This coordinate frame would then allow the robots to run body tracking models selectively based on which feed provides the best view and easily feed it into an intention predictor upon transforming the models' landmarks. The research here covers the setup of the experiment, the process behind the two explored solutions, and the assessment of their effectiveness.

1 Introduction

1.1 Background

Human-robot collaboration has been an emerging field within computer and cognitive science. As robots become more adept at autonomously comprehending and interacting with their surroundings, the scope of their applications and deployment contexts rapidly continues to broaden. Robots have already proven to excel in structured settings when trained to execute well-defined, heavily-conditioned tasks through automation. Due to continued research in this field and the integration of robots into industry, robots are increasingly able to tackle diverse tasks in unstructured settings. This rise in robot autonomy has created more demand for developing robots capable of functioning in different environments and collaborating with humans to achieve common objectives.

Creating robots that can effectively cooperate with humans entails addressing numerous new challenges and complexities. These systems must be able to have a good enough understanding of their environment to anticipate actions, communicate and coordinate efforts with their human partners. These robots can also seriously injure and put the lives of any humans and animals in their surrounding area at risk so they must be capable of prioritizing and ensuring safety. Due to these constraints, it might be a while before we see large-scale robot assistants in our homes or in rapidly changing environments. Nevertheless, there are many environments with enough structure for us to introduce collaborative robots into. The goal of this research project is to build a robot capable of assisting and protecting an electrical technician as they are tasked with diagnosing and resolving faults within a junction box designed to simulate a compact heating, ventilation and air conditioning(HVAC) system. This report focuses on the necessary physical setup for building such a robot and how to leverage different computer vision algorithms to inform the robot about its environment.

2 Experimental Setup

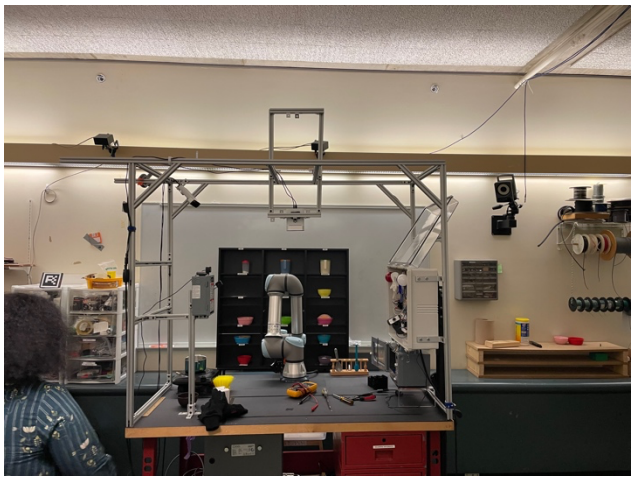

Figure 1: Physical setup

The physical setup is located in Professor Scassellati's social robotics lab on the fifth floor of Arthur K. Watson Hall as shown in Figure 1. The setup is built upon a desk approximately 3 feet above the ground. At the center of this table sits Universal Robotics' UR5e robotic arm. The UR5e is an adaptable collaborative industrial robot that will act as our robot agent and is equipped with a hand-e gripper to

allow it to interact with our environment. To the left, there sits a safety switch that has a lever that will be used to control power to the system.

To the right of the setup sits the junction box that simulates the HVAC system. In it sits a condensed collection of electrical components that one might find within a real heating and ventilation system as shown in Figure 2.

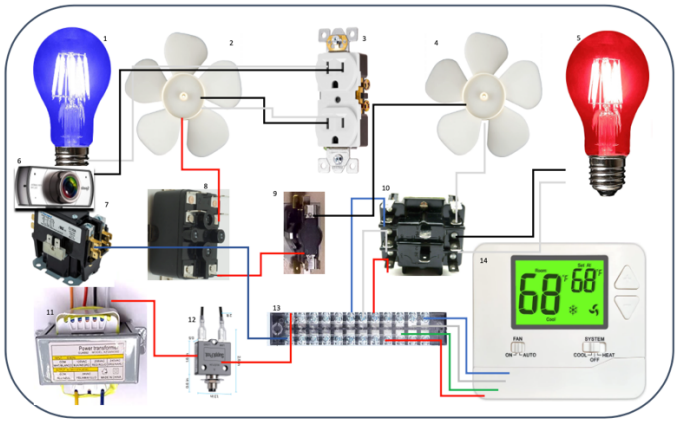

Figure 2: Composition of the junction box

There are four cameras set up around the robot to give the robot system the ability to view and, with the assistance of machine learning models, understand its surroundings. The camera that captures the most information about the environment is the angled camera mounted in the top left corner of the 8020 frames. This is an Azure Kinect from Microsoft that has a view of the electrical technician, the table that holds the tools and a view of the junction box. The view of the junction box is occluded when the robot arm is in operation which necessitates the need for a second perspective. This is where the Logitech camera comes into play as it is positioned above the safety switch which sits directly opposite from the junction box. The other Logitech camera is positioned in the bottom left corner of the junction box and is angled towards the front of the desk to be able to track the gaze of the electrical technician and determine which part of the junction box they seem to be looking at. The last camera is an Azure Kinect positioned in the center of the system and is pointed downwards towards the table we will use to track which tools the electrician picks up to diagnose the HVAC system.

3 Image Stitching

3.1 Motivation

Although both the angled Kinect and safety switch Logitech offer clearer perspectives on the human agent and the junction box respectively, they also capture a lot of redundant information. In order to capture as much information on the human agent and the junction box and its components, this means we would have to run our models for tracking the human's body pose, body movement, head pose and hand movement across both fields. This poses a challenge because to be able to track this information, we would need to run multiple models across both feeds and collect all of that data across both camera feeds. Doing this in addition to controlling the robot and running an intention predicting model that processes all of this data poses a computational bottleneck. This is especially problematic because human collaboration requires

fast data processing for timely robot intervention. In order to lessen the computation load, we decided to try and consolidate both camera feeds to lessen the data recorded to feed into our intention predictor by stitching the relevant information of both feeds and passing that into the model.

Image stitching is a well-studied field within computer vision but the task of combining two images is not trivial. In order to be able to stitch two different images, we have to be able to find similarities between the two frames in question, project one image onto the other based on those similarities and then find a way to blend those two images together[1].

3.2 Feature Detection

In order to find similarities between the two feeds, I explored and implemented different feature detection algorithms. To test the base functionality of my program I used a simplified stitching task using images from the camera feeds as shown in Figure 3.

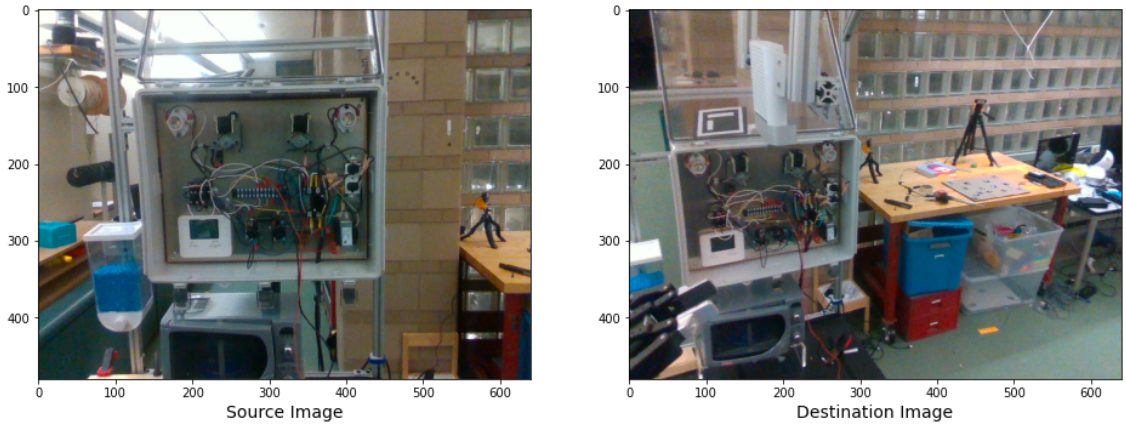

Figure 3: Camera Feed Source and Destination images

The first technique that I explored is using the Scale-Invariant Feature Transform(SIFT) developed by David G. Lowe.[2] It has been proven to be very useful in practice for image matching and object recognition under real-world conditions.[3] This is due to its ability to identify key points and their descriptors invariant to scale, rotation, and illumination changes. It involves five main steps: The first of which is detecting scale-space extrema using a Gaussian pyramid and the difference of gaussians. It then localizes these key points and removes the unstable ones by fitting a quadratic Taylor expansion and discarding low-contrast and edge-like key points. In order to achieve rotation invariance, a constant orientation is calculated using histograms of the gradient magnitude and direction of the pixels around each key point and then a reference orientation is assigned to each key point. The distribution of the directions of the gradients in a neighborhood is calculated again but this time the neighborhood is a circle and the coordinate system is rotated to match the reference orientation. Lastly, the histograms are normalized so that they only store their relations to each other and not the magnitude of the gradients. We are then left with descriptors invariant to illumination changes as well as invariant to the scale and rotation aforementioned. The SIFT descriptor worked well in identifying key points around the edges of objects, especially inside the junction boxes.

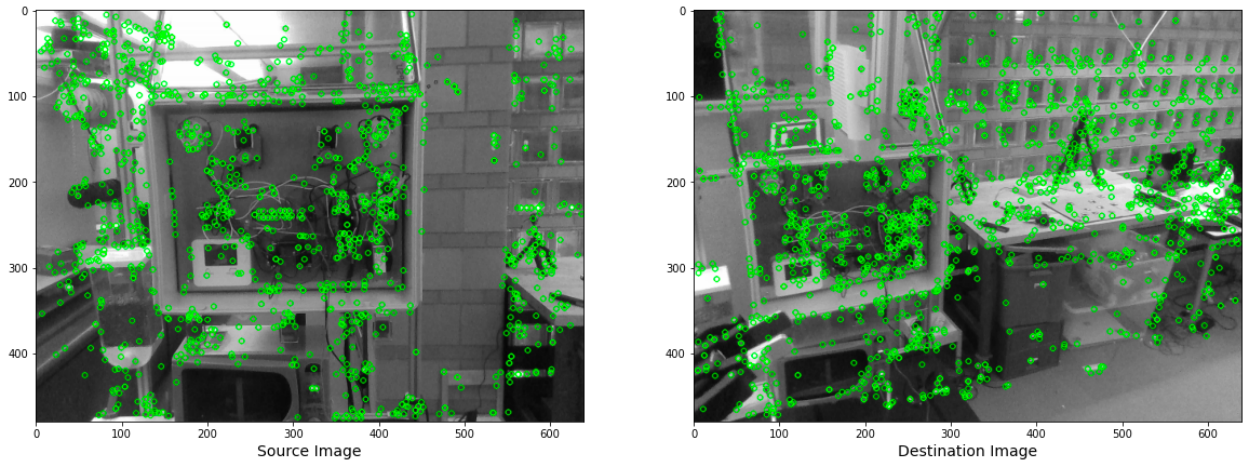

Figure 4: SIFT detector key points overlayed on the camera feeds

SURF is another feature detection algorithm that is popular in object recognition, image registration, classification and 3D reconstruction applications. It was developed by Tony Lindeberg and is partially inspired by SIFT in its model architecture. The use of integral images and box filters in SURF allows for faster scale-space construction and key point detection compared to the Gaussian pyramid and Difference of Gaussians used in SIFT. While SIFT creates a 128-element feature vector for each key point using gradient magnitude and orientation, SURF calculates vertical and horizontal Haar wavelet responses in 32 or 64 elements(depending on whether or not absolute values of the responses are included). While SIFT also assigns each key point a consistent orientation to make them rotationally invariant, this assignment in Haar wavelet responses is optional. Although this results in a rotation-sensitive descriptor, this optional operation and the lower dimensionality of SURF descriptors lead to faster matching and reduced memory consumption.[4] It also retains robustness to scale and illumination changes while significantly reducing computational complexity which I was hoping would make it suitable for this real-time application.

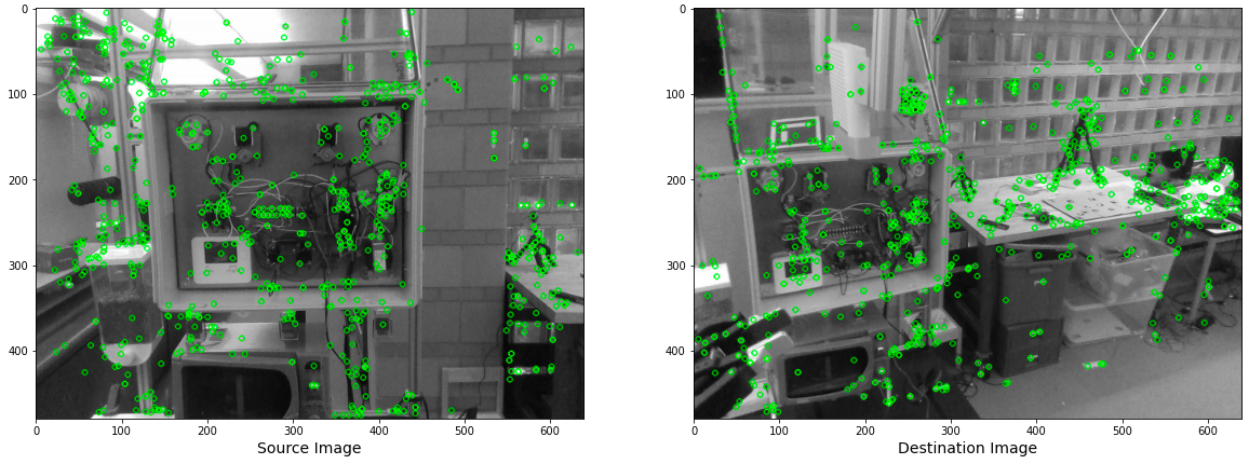

Figure 5: SURF detector key points overlayed on the camera feeds

The same year that SURF was released, another feature detector by Edward Rosten and Tom Drummond also was released for corner detection. FAST stands for features from accelerated segment test and is named after its ability to streamline the corner detection process. It does this by analyzing the circular region around a candidate pixel and classifies that pixel as a key point if there exists n contiguous sequence of pixels around it that are all brighter or darker past a specified threshold. In order to optimize this process, the algorithm optimizes the segment test. It does this by using a decision tree to check the intensity of pixels 1, 5, 9 and 13 and only proceeding if only 3 out of those pixels are above the candidate pixels intensity threshold. This is because the authors found this algorithm doesn't work well if the number of contagious pixels n isn't greater than 12. Although it's estimated to be 300% faster than SIFT, FAST lacks rotation invariance which affects its robustness.[6]

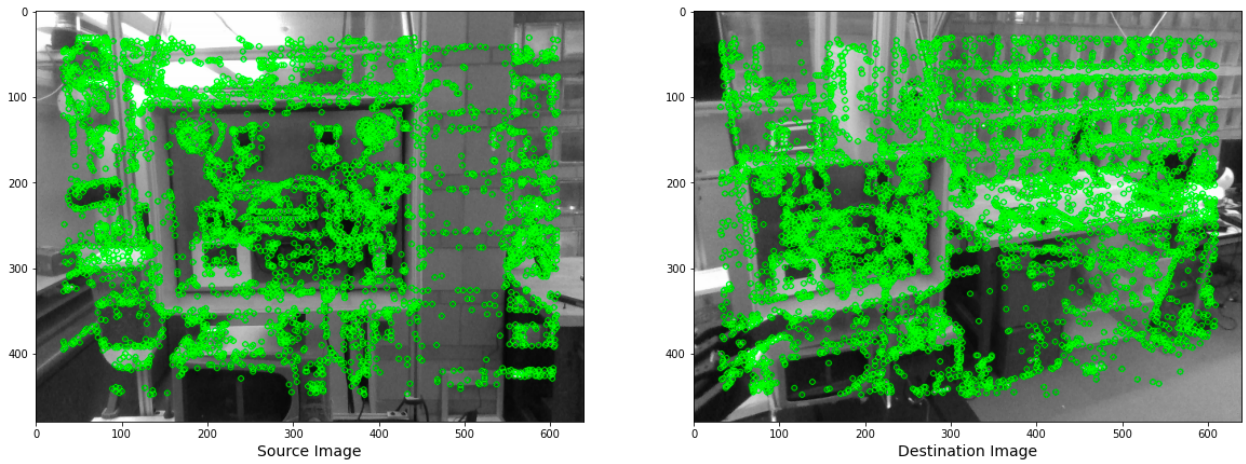

Figure 6: FAST detector key points overlayed on the camera feeds

In order to take advantage of SURF algorithms pace while also being able to detect as many features from the given image, Ethan Rublee created the Oriented FAST and Rotated BRIEF algorithm. The ORB algorithm starts by constructing a scale pyramid of the image using the FAST algorithm to quickly detect features to achieve scale invariance. [5] It selects the key points with the strongest Fast scores and uses image moments to calculate intensity centroids that then provide rotation invariance. The Brief algorithm then compares these pixels with patches around the key point and these pairs are rotated according to their rotation. This ensures rotation invariance and leaves us with binary descriptors which can be compared easily using hamming distance.

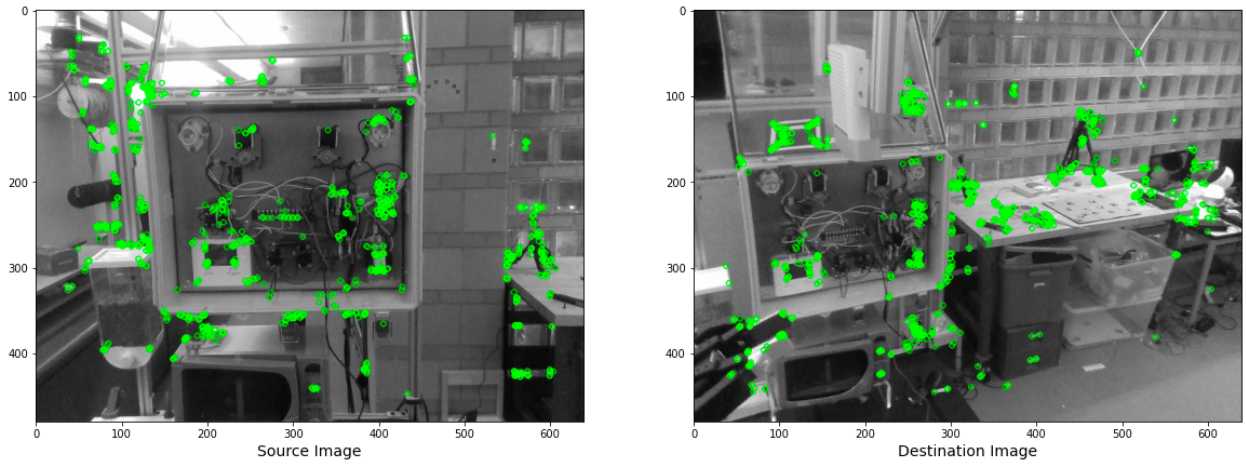

Figure 7: ORB detector key points overlayed on the camera feeds

Table 1: Feature detectors' performance across 200 frames

Feature Detection	Base Image Features Detected as key points	Destination Image Features Detected as key points	Computation time in seconds for both images (in seconds)
SIFT	1689	1313	0.1415
SURF	1174	873	0.6323
FAST	4883	3962	0.0232
ORB	3169	3037	0.0434

3.1.1 Feature Matching

The next step in image stitching is matching the generated key points from the feature detectors across both images. In order to optimize functionality and performance, I explored the most popular feature-matching algorithms which were the Brute-Force matcher and the Fast Library for Approximate Nearest Neighbors(FLANN) matcher.

The Brute-Force Matcher works as its name suggests which is by comparing a descriptor from one image with all the other ones from the other until the list is exhausted. It used Euclidian distance for floating-point descriptors and Hamming distance for binary descriptors to measure feature similarity.[7]

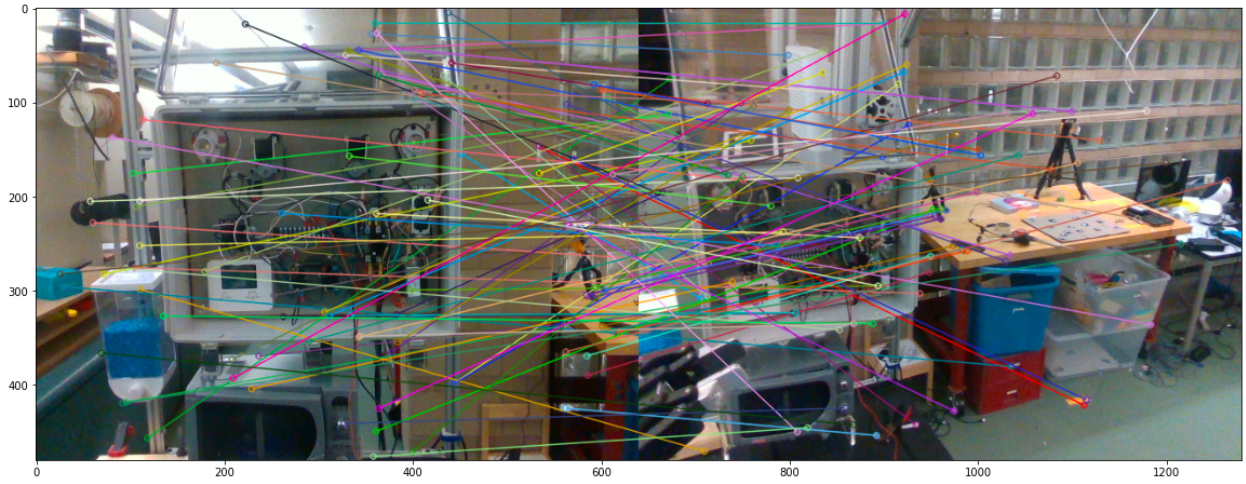

Figure 8: Brute Force Matcher on Sift feature extracted key points

FLANN on the other hand is more efficient as it optimizes the process by using converting the data into nodes grouped in tree-based data structures. It then runs the nearest neighbor algorithm on these feature nodes to approximate the nearest neighbor.[8] Although this makes it faster and finds potential mappings, we do not produce as many feature mappings because we apply the Lowe's test. This test checks if the ratio of the distance between the first and second features is greater than the 0.75 threshold. If that is the case, it is considered a false positive and discarded.[2]

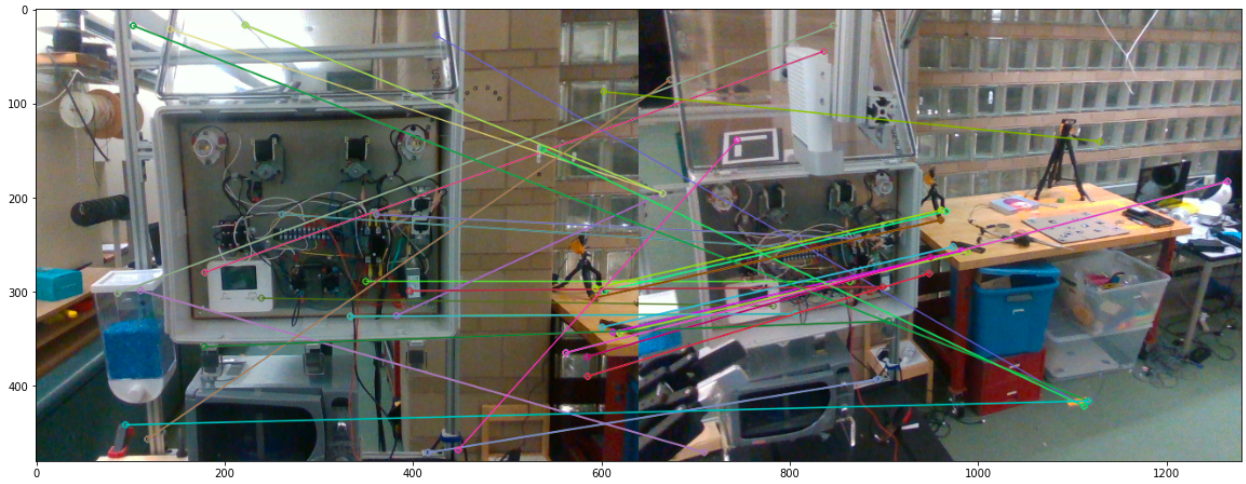

Figure 9: FLANN matcher on Sift feature extracted key points

Table 2. Brute Force(BF) feature matcher performance across 500 frames

Feature Matcher	Number of key point Matches	Computation time (seconds)
SIFT	441	0.025864
SURF	159	0.006730

FAST	1131	0.289646
ORB	868	0.083703

Table 2. FLANN feature matcher performance across 500 frames

Feature Matcher	Number of key point Matches	Computation time(seconds)
SIFT	32	0.0176701
SURF	16	0.0051400
FAST	98	0.1087849
ORB	45	0.0410089

3.1.2 Computing Homography and Image Blending

Once we have generated the feature mappings, I needed to calculate the homography matrix using the Direct Linear Transformation algorithm to transform the points from one image to another. There are 8 degrees of freedom in a 3x3 homography matrix and since each match provides 2 constraints in its x and y coordinates, it takes at least 4 matching points to be able to generate this matrix. The feature mappings are formed into a linear system of equations and using Singular Value Decomposition(SVD), we can calculate the homography matrix.

When we have more than 4 correspondences, we tend to have outliers which are mappings of points that have similar features but don't actually correspond to the same point in the real world. SVD doesn't deal well with outliers which means that this algorithm is usually paired with another algorithm like Random sample consensus(RANSAC) to remove them. [11]

RANSAC works by iteratively selecting a random subset of feature matchings, estimating the homography using DLT and then measuring the alignment of all the other feature matches. The algorithm stops once a defined percentage of feature matches are aligned or the RANSAC algorithm reaches the end of a predefined number of iterations and the homography matrix with the highest portion of matches is selected.

Once we have the transformation matrix, we now know which points from our source image correspond to which points in our destination image. Before transforming the source image, we pad the destination image with empty pixels to expand its dimension so that we don't lose any information. In order to avoid the presence of seams at the points the images are stitched, we also have to account for exposure differences. By using mean value blending which just means assigning a pixel value the average of the base image and the transformed image, we can generate a seamless stitched image.

3.2 Results

Although the FLANN matcher produced fewer point matches as expected, there were not enough key points that were appropriately matched in order to render appropriate transformations. All of

the FLANN images came out with distorted rays as shown in the BF outputs of SURF FAST and ORB in Figure 10. The Brute Force matcher for SURF on the other hand, which was able to capture many key points invariant to scale, illumination and rotation was able to render a stitching that had a lot of defects.

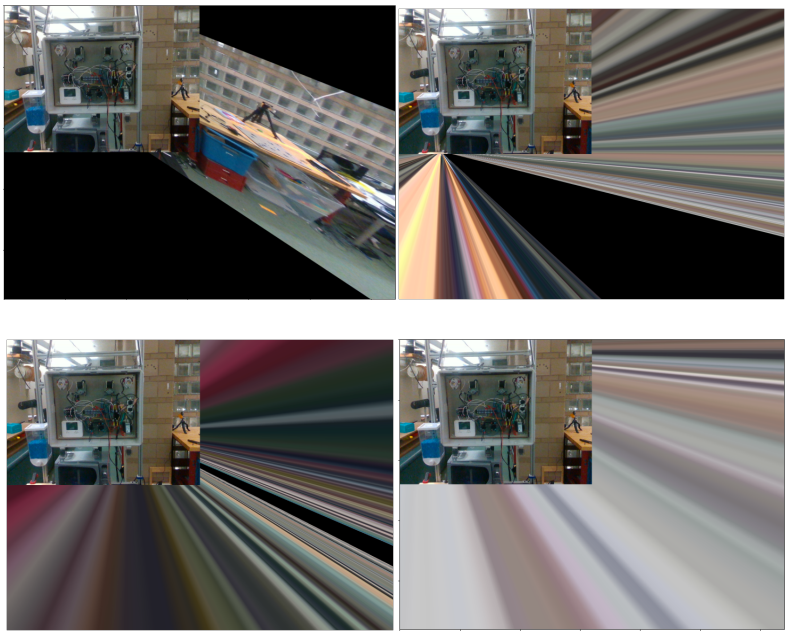

Figure 10: BF matcher renders (Top row: SIFT and SURF, Bottom row: FAST and ORB)

Although changing the number of key points rendered did lead to small improvements in the renders of the surf matcher, the angle and difference in focal point height of the cameras was too different for any useful stitching across all models. In order to fix this, I realized the cameras need to be closer to one another. There is a camera already pointed at the desk that can be used to gather information about the tools being used so I experimented with different found camera positions to find what worked best. Images stitched from close to the same height seemed to perform best in this stitching task so I found a configuration that allowed for that while still being able to see the junction box and the person. Afterward, I installed a horizontal 8020 bar onto the UR table frame and mounted two Intel RealSense D435 Depth Camera's as shown in Figure 11.

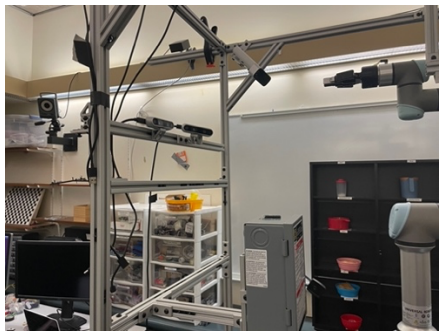

Figure 11: new camera setup and their corresponding FOV

Running the feature detectors and matchers on the updated camera location significantly improved the outputs for the image stitcher. Although the number of key points and computation time remained similar, the feature matchers performed much better. Due to the lack of significant rotation, I was able to use the FAST feature detector and approximate matches using the FLANN detector to generate the stitchings shown in Figure 12.

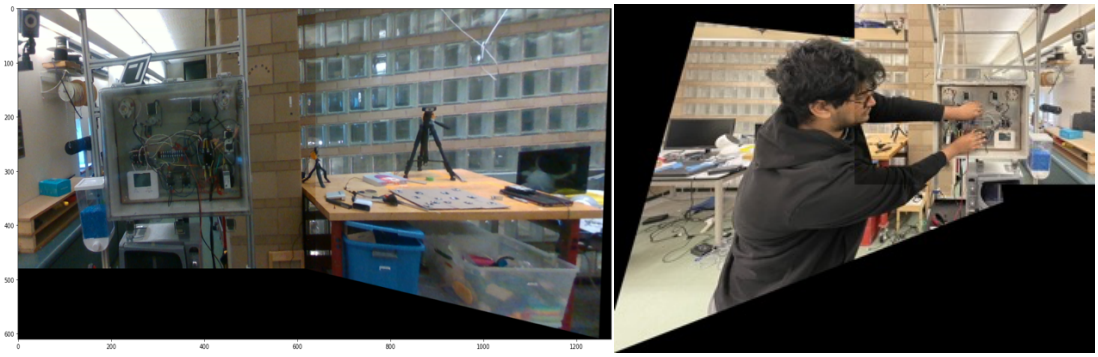

Figure 12: Image stitchings with new camera positions

After generating a suitable image stitcher for the environment, I wanted to test how it performed on frames where we have a human agent interacting with the junction box. After saving the feeds of a human agent and his hands testing the components in the box using a digital multimeter, we stitched them together. We were then able to test the performance of these stitched images by running it through Google's MediaPipe's body pose tracker and hand detector. We then overlayed those landmarks over the stitched image as shown in Figure 13. You can see from that image that the body pose model runs seems to map correctly but the left arm vectors and hand are not being picked up by either model.

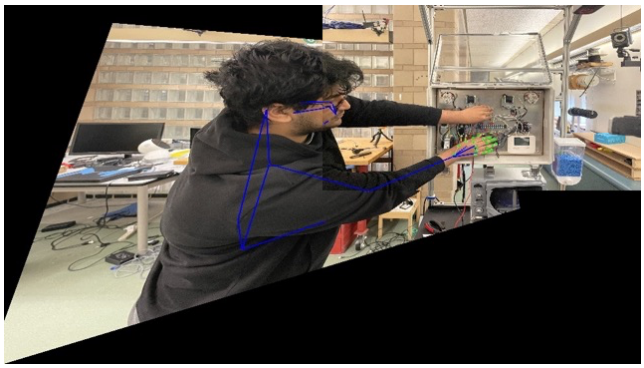

Figure 13: Stitched image of Human agent with Hand tracker and Body Pose models overlayed

4 Coordinate Frame Transformation

4.1 Aruco Marker

Another potential solution I explored for optimizing the model load sharing was to try establishing a universal coordinate frame for the two camera feeds. This would allow for different machine learning models to track the environment and the human using the same relative coordinate frame to then feed into the intention predictor. In order to implement this, I used aruco codes to obtain experimented with aruco codes.

Aruco codes are a type of fiducial marker that have an internal binary pattern used to encode an identifier. These codes are arranged as white square pixels on a black background and the contrast allows them to be detected from far away and in any orientation. The OpenCV aruco library handles all the detection and identification for us but in order be extract its pose, I needed to capture the realsense camera's intrinsic matrix(its focal length and principal point) and the camera's distortion coefficients using the realsense Robot Operating System(ROS) camerainfo topic. I then fed these matrices into OpenCV's aruco pose detector which will give us the translation vector and rotation matrices. We can then calculate the rotation matrix for a 4x4 affine transformation matrix by taking the dot product of the two aruco pose rotation matrices. To get the translation vector, we first compute the inverse transformation of the first marker which is the dot product of the second markers rotation matrix with the inverse transformation of the first marker which then gets added to the second markers translation vector.[10]

The cameras also tend to move around a lot so I created the calibration board shown in Figure 13 based on the relative placements of the junction box components.

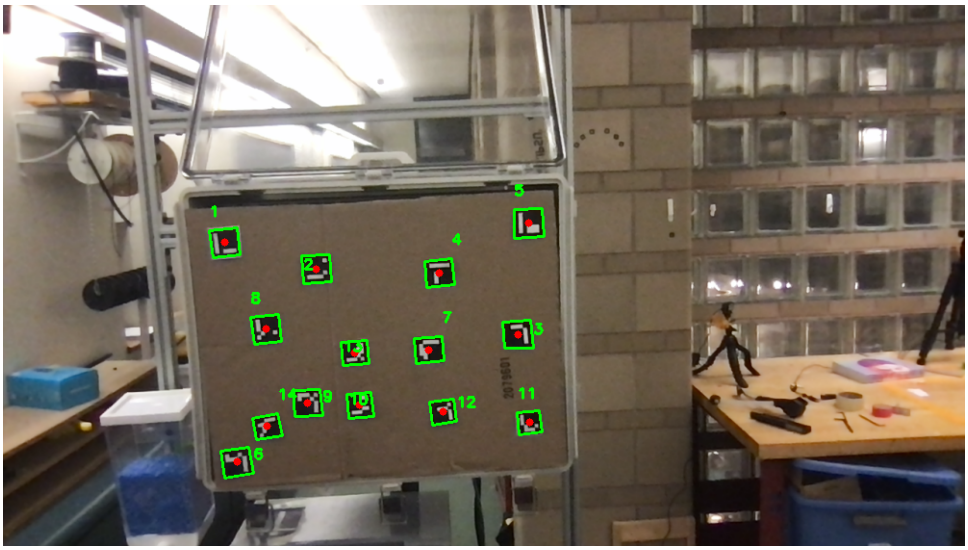

Figure 13: Calibration board with aruco codes with unique identifiers listed here. (1) Blue Bulb (Compressor), (2) Condenser Fan, (3) Dual Plug Outlet, (4) Heater Fan, (5) Red Bulb (Heater), (6) Gaze webcam, (7) Contactor Relay, (8) SPST Fan Relay, (9) Electric Heat Sequencer, (10) DPDT Switching Relay, (11) Power Transformer, (12) Circuit Breaker, (13) Terminal Block, (14) Thermostat with Display

5 Conclusion

The image stitching approach seems promising for consolidating the necessary camera feeds with drawbacks in performance in the current environment. In order to be able to track a human using these stitched images, it would require getting cameras with a better RGB feed resolution and no distortion across the two camera feeds. The camera would also need to be placed at around the same height in order to be able to use off-the-shelf body and face tracking models. In addition, the feature detection, matching and transformation process took approximately a second to generate on the UR computer, which has a AMD® Ryzen threadripper 2950 16-core processor. This significant time delay for processing the images, the time required for running the necessary models on the output image and then the time necessary to feed those points into the intention predictor is not conducive to real-time feedback from the robot.

The transformation coordinate system also has significant translation errors in its current state. I keep running into erroneous outputs as to which component was being touched when placing a coordinate in one frame and transforming the coordinates of that feed to the other feed's coordinate frame. Figure 14 shows an example of these points being overlayed the feeds with the aruco code frame and position also overlayed onto the markers. These miscalculations sometimes accounted to over 8 inches in junction box target which is not viable for a junction box so densely packed.

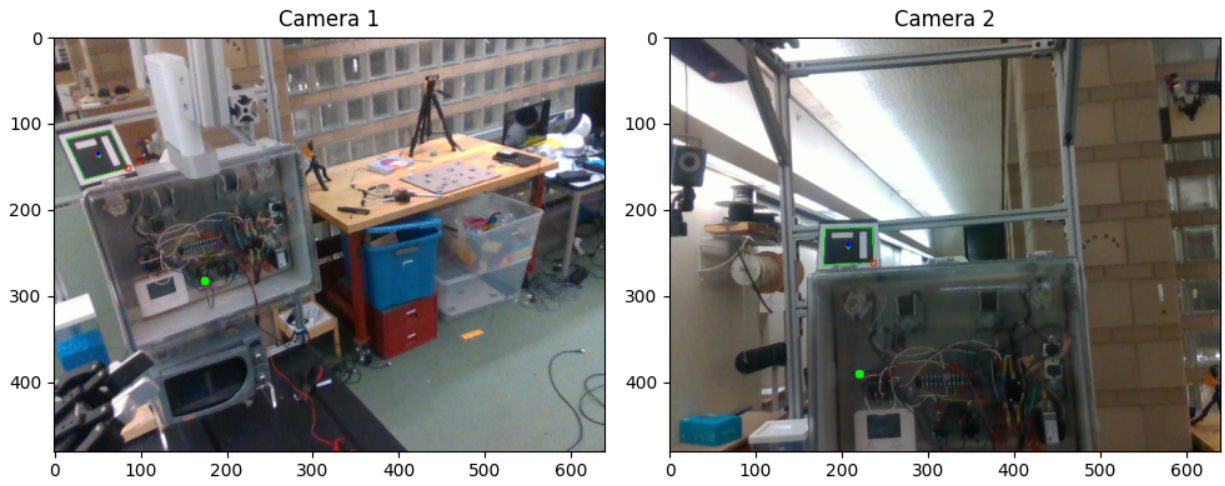

Figure 14: Point projection across the two camera frames

6 Hardware Modifications

6.1 Arduino and Relay Accessibility

Throughout this project, I also worked to help develop the fault-triggering system that is controlled by an Arduino that sends signals to the solid-state relays on the back. Although the specific erroneous scenarios are still being developed, I 3D printed and installed a plastic handle and standoffs to make a sliding door on the back of the junction box. This allowed us to diagnose and debug errors within the safety system discretely while making sure that human participants are not aware of the fault-triggering mechanism during experiments.

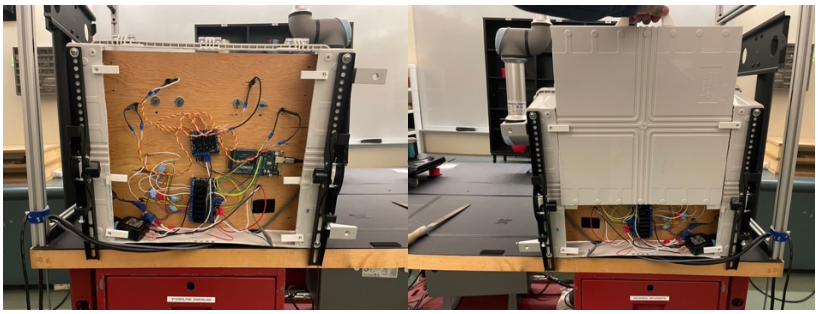

Figure 15: Modified junction box

6.2 Safety Improvement

We also installed a ground fault circuit interrupter(GFCI) for IRB purposes and also to protect our Electrical technician's when we run user studies. The GFCI will sense when the amperage flowing into a circuit differs from the amount flowing out. It then turns off in less than one-tenth of a second if it senses a difference as small as 5 milliamps.

7 Future work

The image stitching approach with our current cameras does not seem to be a great real-time solution, but we have been concurrently developing the program that has all the models running on both cameras. Once we have designed and built the intention predictor model, we hope to test the extent to which the size of the data stream affects the delay in the models output. We also hope to explore other ways of optimizing this process like parallelizing the computation process by having the different camera feeds being processed on different computers and then communicating relevant information with each other through the same ROS network.

I'm also hoping to fix the aruco code coordinate system transformation process by using charuco codes to calibrate the camera parameters instead of getting the values from the cameras ROS topic. This will hopefully fix the transformation errors to better asses the viability of selectively pairing specific models with the camera feed that has the best view of that part of the body. I am also hoping to test the image stitching performance again by utilizing the computer's GPU to cut run time to better test its viability in this application.

8 Acknowledgments

I would like to thank Professor Scassellati as well as Debasmita and Emmanuel for allowing me to join this project and providing guidance. I would also not be here without my family especially my mother and older siblings Bruck and Feven. Lastly, thank you Joel for being my human model and A42 for being my support system throughout these 4 years.

References

1. *Richard Szeliski. 2006. Image alignment and stitching: a tutorial. Found. Trends. Comput. Graph. Vis. 2, 1 (January 2006), 1–104. https://doi.org/10.1561/0600000009*
2. Brown, M., and Lowe, D. G. (2007). Automatic Panoramic Image Stitching using Invariant Features. International Journal of Computer Vision, 74(1), 59-73.
3. Lindeberg, Tony. (2012). Scale Invariant Feature Transform. 10.4249/scholarpedia.10491.
4. Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features." In European Conference on Computer Vision, pages 404-417, 2006.
5. Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An Efficient Alternative to SIFT or SURF." In International Conference on Computer Vision, pages 2564-2571, 2011.
6. Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection." In European Conference on Computer Vision, pages 430-443, 2006.
7. Jakubovic, Amila & Velagic, Jasmin. (2018). Image Feature Matching and Object Detection Using Brute-Force Matchers. 83-86. 10.23919/ELMAR.2018.8534641.
8. M. Muja and D. G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." In International Conference on Computer Vision Theory and Applications, vol. 1, pp. 331-340, 2009.
9. Jos´e Luis Blanco Claraco. (2022).A tutorial on SE(3) transformation parameterizations and on-manifold optimization
10. HARTLEY, Richard, and Andrew ZISSERMAN. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2011.
11. Fischler, M. A., & Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Communications of the ACM, 24(6), 381-395.