A0: AutoPuzzlr

Authors: Andrew Conduff, Connor Maggio, Aneek Mukherjee: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—AutoPuzzlr is an automatic puzzle-solving system that guides users as they work through a puzzle to speed up the sometimes lengthy process. We are using modern technology to allow our users to reap the mental benefits of solving puzzles while reducing the time committed and easing the difficulty, while also offering the benefit of ensuring that all pieces are accounted for. This is a niche and relatively untouched space commercially and no competing technology can claim the features that AutoPuzzlr will have, so we are pioneering a much more advanced and capable product for passionate puzzle solvers!**

***Index Terms*—Computer Vision, Feature Matching, Hand-tracking, Leap Motion, Projector, Puzzle, Solving, Tap Detection**

I. Introduction

AutoPuzzlr is a project that is designed to help a user complete a puzzle using modern technology and an intuitive user interface for the user to interact with. Doing puzzles offer many benefits for their users, but sometimes these benefits can be hard to reap due to the time it takes to complete a puzzle as well as the difficulties puzzles can bring. Furthermore, the disappointment of finding out that there is a piece missing from the puzzle only after completing the rest of the puzzle can be minimized because of the capabilities of this project to rapidly count pieces. Given the niche area, there are few other competing technologies, and no other technology claims to be able to handle the same set of features this project is able to boast. This project can claim a more technologically advanced solution. (e.g. To count many pieces, a common strategy is just to weigh the total number of pieces and create an estimate). The goals of this project are to be able to guide a user to build a puzzle through a touch interface on the physical puzzle itself. This system, from user tap of a piece to display of where that piece should go in 4 seconds, for any piece within the workspace. This will be achieved through the use of a Leap Motion controller, a Logitech C920 webcam, and an Epson Powerlite projector all controlled by a Raspberry Pi Model 4. This project should achieve a 90% accuracy for piece placement within half an inch of the final piece placement.

II. Design Requirements

Our high-level user requirements are as follows:

- End-to-end suggestion latency: 4 seconds to provide a suggestion to the user
- Suggestion Precision: .5 inch between piece's suggested and actual location
- Suggestion Accuracy: 90% of the time the piece will satisfy the precision requirement

We understand that we need to be able to account for some errors in the environment itself and that no computer vision code will be perfect so we thought that having a 90% suggestion accuracy would be a high enough placement accuracy score such that the user can rely on it, but if the piece does not fall within that half inch radius of where that piece should go, then that would be considered an inaccurate placement of a piece. Our design is robust for detailed pieces, but given that some puzzles have similar textures across wide swaths of the picture, there will be a circle of confidence of where that piece could be that will grow larger across similar areas. (e.g. in a puzzle with a lot of open sky with a lot of blank sky-blue pieces, then our accuracy will likely be a lot lower than a puzzle piece with a specific detail on it.) We chose a 4 second design time because of the limitations of technology and the algorithms we are using. We decided that 4 seconds was a tight enough constraint such that it would still feel intuitive and useful to a user, but gave us enough time to compute where these pieces should go.

We have outlined that there is a 4 second response time between user input (tap) and the display of the projector onto the surface. We have further subdivided this into the following list:

- Tap Detection: 50 milliseconds from actual tap to tap notification in the back-end
- Piece Identification: 500 milliseconds from coordinates provided to CV system to identified piece
- Piece Matching: 3 seconds from identified piece to suggested location returned to back-end
- Response Latency: 50 milliseconds from a user tap notification or returned coordinates in the back-end a response graphic will be displayed.

Our timing requirements are derived from estimates of computation power required for the algorithms that we are using as well as our limited computation power of the Raspberry Pi. Our 50ms timing requirement for tap detection is derived from the amount of time for a tap to be seen as highly responsive for human perception.

Our hardware performance metrics are as follows:

- Camera Field of View: 100% of our ~24"x24" workspace
- Camera Sensor: 12+ MP resolution and good color identification
- Projector Image: 100% of our ~24"x24" workspace

- Projector Brightness: Sufficient to be visible against workspace in varied light conditions

These requirements are based off of the size of the puzzle and we have an upper limit of about 20” by 20” for our maximum puzzle size. We came upon the 12+ MP resolution camera as this should be sufficient given the distance of the camera as well as the average size of puzzle pieces. The least technical of these requirements would be that of the projector’s brightness as it will be environmentally dependent, and that is up to the user’s light sensitivity. Some users may prefer a very bright projector light in a very dark room whereas others may want a softer display in a brighter environment.

III. Architecture and/or Principle of Operation

Our system architecture is described visually in Fig. 1.

Our system is composed of a PVC frame containing all of our components and defining the workspace for the user. The frame will hold the hardware for the system - a webcam, projector, hand-tracking sensor, and a Raspberry Pi for computation. These components were chosen to satisfy the user interaction requirements we set and we discuss the selection process in the following Section (IV).

The software system is broken down into 4 libraries functioning like microservices - the CV system, the tap detection, the animation & display builder, and the back-end. The former three components each represent a core system operation in the user’s interaction cycle with the system.

The user’s taps are first recognized by the tap detection and localization service, which is triggered by the back-end to start looking for taps. Only the coordinates of the tap are reported back via local socket.

Then, the CV system takes the remapped coordinates from the back-end and identifies the piece, runs its feature and orientation matching algorithm, and reports the location of the suggested final location, as coordinates, back to the back-end.

Finally, the animation & display builder service takes the tap and suggestion coordinates from the back-end and creates and outputs animations to the user via the projector.

Our system is architected in this way so that there is clear separation of responsibilities between components and a high level of possible parallelization in the development process, since there is a clear API between otherwise independent components. It also allowed us to thoughtfully create the API so that minimal data is being transferred to help keep our latency down.

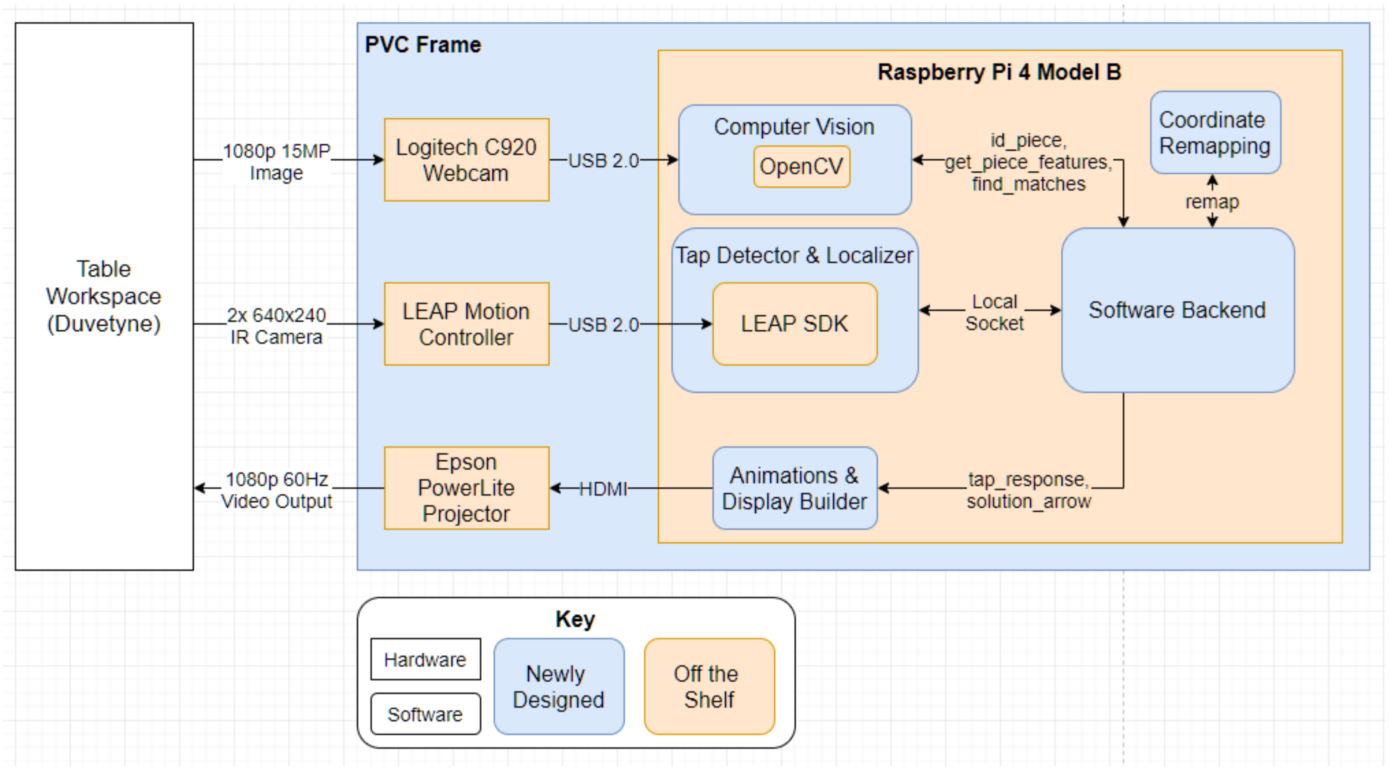


Fig. 1. A complete block diagram of out system architecture

IV. Design Trade Studies

A. *Physical Components*

Our project requires a camera, a projector, and a frame. We have specific functions for each of these components and carefully considered our options.

1) *Frame*

Our frame was designed keeping in mind that it needs to be portable, adjustable, and relatively cheap. We decided to use 1.5" diameter Schedule 40 PVC due to its sturdiness, light weight, and price compared to wood. We decided on a 1.5" diameter because of its rigidity and reasonable weight. We will need to drill screws into some of the PVC connections in order to mount the projector. This diameter leaves room for this.

In addition, PVC piping has reliable connectors that include sliding components. We may need to adjust the heights of our projector or cameras in our prototyping efforts. Using PVC makes this easily possible.

Our design is susceptible to toppling over from the weight of the projector. We considered preventing by adding a counterweight to the base of the frame or simply adding an extending pipe to the base. Adding an extension would have an effect on the user's workspace. Thus, a counterweight was a better option.

2) *Surface*

The work surface cannot reflect too much light as this affects the Leap Motion controller's performance. We considered making the work surface clear acrylic or using a light absorbing fabric. We decided we would like to avoid adding legs below the surface. Thus, we will instead be covering the surface of the table with a sheet of Duvetyne fabric, which absorbs IR light and provides an even dark background. Further discussion of this decision is contained in Section IV.C.5.

3) *Camera*

We considered a few options for our camera. Initially, we were debating whether to use a DSLR, a smart phone camera, or a personal web camera. Personal web cameras are the best choice for our project's scope. That is to say, a DSLR camera or a smartphone have far too many functions that would go unused for us.

The web camera we decided to use is the Logitech C920. In choosing this, we mainly considered camera sensor size, megapixel count, and price. Thanks to CMU's IDeATe and ECE departments we had lending access to a couple models including Logitech's C615 and C920 webcams, and Quickcam's Pro 9000 model. While each camera had a similar sensor size, the C920 has the largest megapixel count at 15 MP. Upon further testing, we decided that we would require this resolution to ensure our image recognition requirements. Past projects that have used OpenCV have succeeded using this camera.

4) *Projector*

Our projector was chosen based on size, throw ratio, lumens, and price. We would have liked to use a mini projector in order to cut down on weight, however they tended to have a low number of lumens. Since our product will be used in the light, we required at least 2500 lumens. In addition, we were looking for a high throw ratio in order to meet our requirements of projecting onto a 20" by 20" puzzle within a distance of 4 ft. The Apeman M7 mini and the Epson Powerlite 1776W were the only models that met the lumens and price specifications, as the Apeman was quite affordable and the Epson Powerlite was already owned by the ECE department. These and other projectors considered are shown in Fig. 2. Upon testing, we found that the Apeman projector projected a very wide screen that would limit us down to around 16" puzzle heights. The Epson Powerlite 1776W was the clear choice moving forward.

5) *Computer*

Our computation currently is being developed on a quad core i7 with a 2.6 GHz processor. We are hoping to use a Raspberry Pi in order to localize the entire project onto our frame. However, the Raspberry Pi has a quad core with a 1.3 GHz processor. If this does not meet our timing requirements upon testing, we will likely revert the computation back to one of our laptops, or offload to an AWS GPU instance, based on our latency measurements at the time.

B. *Computer Vision*

Our computer vision relies heavily on a pipeline of image processing techniques, however we need to balance both the speed of these applications as well as how robust they are, and there is still a lot of tweaking to be done as we get the parts we need and see how everything translates computationally to the Pi.

1) *Technologies Used*

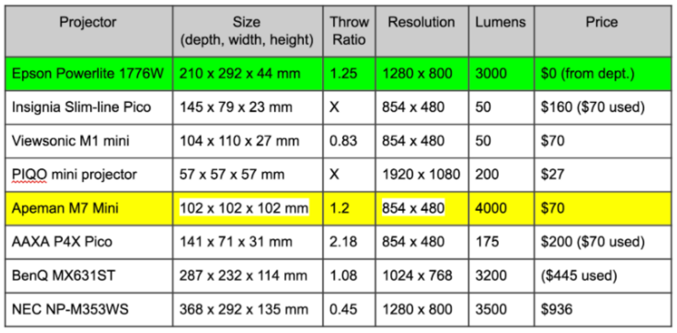
We have decided to use the Python version of OpenCV version 4.2.0, as it was the most up to date version at the start of our project and the OpenCV community is vast and helpful in case we ran into any problems.

2) *Thresholding*

We have tested many methods, but for now we are planning on using OpenCV's built in THRESH_BINARY_INV method for thresholding, in combination with THRESH_OTSU, in order to increase the confidence of our thresholding. (See labeled picture #2 in Fig. 3). We felt like this gave us the best combination of background filtering as well as the ability to clearly choose what part of the image is the puzzle piece. Fig. 3 picture 4 preserves the picture, but that is not important for this step of our algorithm. Once we have this piece separated, we are able to extract only those pixel values and consequently detect features across them.

3) *Feature Detection*

While we could go with methods like SIFT and SURF we



Projector	Size (depth, width, height)	Throw Ratio	Resolution	Lumens	Price
Epson Powerlite 1776W	210 x 292 x 44 mm	1.25	1280 x 800	3000	$0 (from dept.)
Insignia Slim-line Pico	145 x 79 x 23 mm	X	854 x 480	50	$160 ($70 used)
Viewsonic M1 mini	104 x 110 x 27 mm	0.83	854 x 480	50	$70
PIQO mini projector	57 x 57 x 57 mm	X	1920 x 1080	200	$27
Apeman M7 Mini	102 x 102 x 102 mm	1.2	854 x 480	4000	$70
AAXA P4X Pico	141 x 71 x 31 mm	2.18	854 x 480	175	$200 ($70 used)
BenQ MX631ST	287 x 232 x 114 mm	1.08	1024 x 768	3200	($445 used)
NEC NP-M353WS	368 x 292 x 135 mm	0.45	1280 x 800	3500	$936

Fig. 2. A table of various projectors we considered

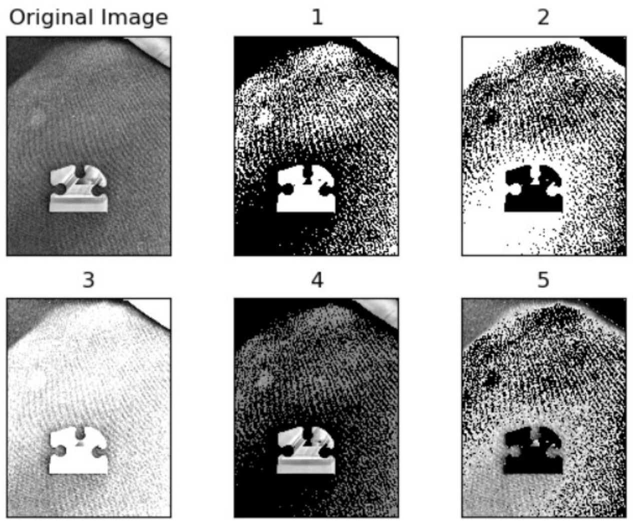


Fig. 3. An image of a puzzle piece after various thresholding operations

didn't feel like it was necessary as they are not open source in all cases. We have opted to use the ORB method as it is not only comparable to those methods, but also free and this means that we will not have to write these functions, adding more time to what would be our project.

4) Background Cloth

We have decided to use Duvetyne fabric as the base of our operations as compared to table surfaces (glossy wood and glossy capstone benches) as well as my laptop sleeve, we have opted to get a fabric which is known to vastly limit these kinds of reflection and should stop the creation of many of the artifacts and noise seen in the background of the previous photo.

C. Tap Detection

We wanted user interaction with our service to be as close to the reality of solving a puzzle, so we are utilizing the Leap Motion controller to identify users' hands and taps in the workspace. This removes a hardware layer between users and their puzzles and allows us to get user input in an unobtrusive way. Some key design decisions for this subsystem included choice of SDK, technologies used, API design, orientation of the controller, and the surface of the workspace.

1) SDK

The Leap Motion controller was originally designed for using as a touchscreen/keyboard replacement or creating a virtual reality control surface. However, the company has since pivoted toward Virtual Reality applications and their recent libraries are exclusively for Unity/Unreal Engine. To use the latest SDK with a Python program, we would have to spin up a VR project on the Raspberry Pi. Therefore, we are utilizing an older Python SDK rather than the latest version so that we can avoid the unnecessary and significant computation overhead of running a VR project.

2) Technologies Used

Our decision to use the older SDK constrained us to Python 2.7 for interacting with the SDK and controller. However, we wanted to take advantage of the modern features and performant libraries available to us in Python 3.8.1, so we decided to separate the entire Tap Detection & Localization component into a separate process that communicates with the back-end via local socket. This decision allows us to separate the Python 2.7 code while still maintaining minimal latency by using local sockets. We decided that since these modules were part of the same program and would be running concurrently, we will be setting up the socket on initialization and closing it on teardown, which will greatly simplify development.

3) API Design

Since we are communicating with the back-end via local socket, we needed to define an API so that we can minimize data transferred and have a clear understanding of how both sides will communicate to support parallel development. We decided to have a call-and-response style of system, where the backend sends a trigger, and the service responds with the data on the following detected tap.

We considered alternatives, such as constantly running detection and reporting every found tap, but this approach required the back-end to continuously monitor the received messages to make sure that it responded to the tap that the user intended to make when they are prompted by the system to tap a piece, rather than any randomly recognized taps made while computation on a previous piece was occurring. It made the most sense to only report a detected tap when the system was expecting one.

4) Orientation

Due to the original design goals of the Leap Motion controller, the software was optimized for using the controller face-up on a surface, tracking hands *above* it. Further research also showed that this older SDK included optimizations for tracking *palms*, since it assumed the upward-facing orientation. This orientation is not possible for our purposes, since the user will be tapping puzzle pieces directly on the work surface rather than tapping the air above them, so we experimented with the hand tracking under viable orientations - mainly, mounted overhead facing down, and mounted on a vertical column and facing sideways.

We utilized software from Leap Motion to view the actual IR camera inputs to make these observations. We realized immediately that the tracking was extremely poor when oriented sideways, as most of the hand is blocked and only the side of a palm is visible to the camera. This proved to be nearly impossible for the tracking software to recognize, with the palm-tracking optimizations enabled or disabled. Overhead tracking was better, but the effective range was far lower than claimed in the data sheet. We were able to track hands up to approximately 10", while the datasheet claimed two feet. One issue was certainly that our use case necessitated that the controller track the backs of users' hands rather than the palms, but another issue was the work surface itself, which we discuss in the following section. Regardless, the downward-facing orientation is clearly the better option, and that is the orientation we decided to move forward with.

5) Work Surface

We had determined that the downward-facing orientation was our best option, but we hadn't yet been able to reach the performance levels we were looking for and were promised in the device specifications. We continued to use the visualization tool and compared performance between the down-facing and standard, upward-facing orientations. We observed that the primary difference in the images between the orientations was the contrast of the hands against the background in the camera input, as shown in Fig. 4.A and Fig. 4.C. The surface below the hands was reflecting IR light back into the camera and washing out the image when the controller was down-facing, but there was no such reflection in the up-facing orientation and any hands in-frame were clearly contrasted against the background.

We tested this hypothesis by holding the controller 6' above the ground and tracking hands 4'-5' above the ground. The distance to the ground would ensure that little to no IR light was reflected back into the camera and would accurately simulate the upward-facing orientation. With that setup, we were able to achieve the 2' tracking distance that we were looking for, and looking at the image in the visualization tool, shown in Fig. 4.B, confirmed our hypothesis that the reflected IR light from a nearby surface was causing the tracking issues.

We researched mitigations and discovered that an IR absorbent material covering the workspace would increase the contrast of the users' hands and still allow them to work on a surface rather than 4' in air. We found a few options, including Aktar foil and Duvetyne fabric. Aktar was an order of magnitude more expensive than Duvetyne ($199 and $17 for comparable amounts, respectively), so we decided to use a Duvetyne sheet to cover the work surface under the frame and to provide the necessary contrast for the Leap Motion controller.

D. *Software Back-end*

The software back-end is crucial to orchestrating the different components of the system and the key design decision made to fit our requirements and the requirements of each other service was the choice of language used.

1) *Language Used*

Taking into account the skills and experience of the team, our primary language options are Python and C/C++. There are definite performance benefits to using C/C++, since it is a compiled language. However, we decided to use Python 3.8.1 (the latest version) for the bulk of this project because it is much more familiar to us and thus allows for a higher speed of development. Additionally, the OpenCV implementation in Python uses compiled C++ under-the-hood, so we are able to utilize the benefits of compiled OpenCV and use Python.

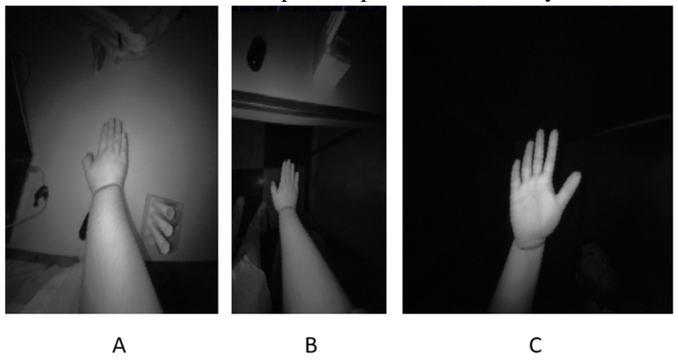


Fig. 4. Images of the camera input from the Leap Motion controller

V. System Description

A. *Physical Components*

1) *Frame*

The frame will be made of PVC and hold all the hardware components. Its base defines the work surface for the user to complete the puzzle within. From the base, two horizontal bars providing mount points extend upward at variable heights so that the resolution can be fine-tuned after construction. One mount point will be at approximately 2 ft., and the other at approximately 3 ft. The PVC opposite the higher mount point will be filled with sand or a similar material to provide a counterweight for balance. The frame is displayed in Fig. 5.

2) *Surface*

The work surface will be covered by a sheet of Duvetyne fabric, which the frame will sit on top of. Duvetyne absorbs IR light and will provide an evenly dark background for the webcam, so it will improve the accuracy of both the tap detection and the computer vision systems.

3) *Computer*

Our software will run on a Raspberry Pi 4 Model B mounted upward-facing on the lower mount point of the frame, along with the webcam and Leap Motion controller.

4) *Camera*

The Logitech C920 webcam will be mounted downward-facing on the lower mount point of the frame. It will be connected to the Pi via an attached USB cable and centered over the work surface.

5) *Projector*

The Epson PowerLite 1776W projector will be mounted downward-facing on the higher mount point of the frame. It will be connected to the Pi via an HDMI cable and offset to the edge of the work surface, since it projects at an angle.

6) *Leap Motion Controller*

Our Leap Motion controller will be mounted downward-facing on the lower mount point of the frame. It will be connected to the Pi via an included USB cable, and slightly offset toward the side of the frame closer to the user, as their hands will likely be working in the space closer to them rather than at the far edges of the work surface, and this positioning will improve accuracy in the most-used space.

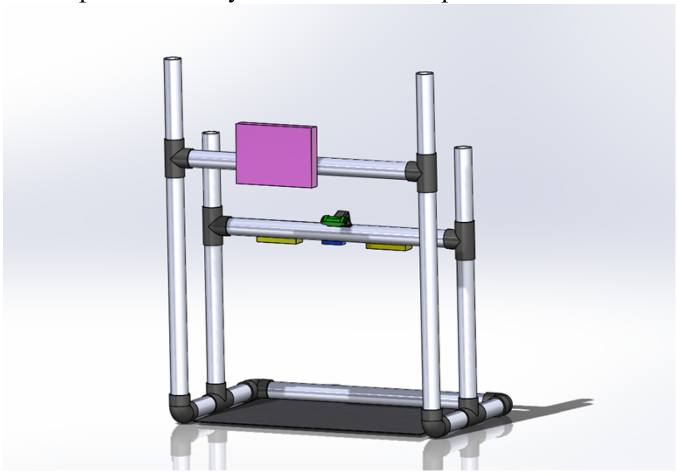


Fig. 5. An AutoCAD drawing of our frame design

B. *Computer Vision*

Our computer vision relies heavily on a pipeline of image processing techniques.

1) The Tap

While not purely computer vision, this sets off the pipeline. This is discussed further in the following Section V.C.

2) Segmentation

Once we have resolved the user's tap location, through our software back-end's coordinate re-mapping, we are able to determine the exact piece that is tapped by the user, and segment that piece out away from the background. This is accomplished through a variety of thresholding techniques as well as some other effects to reduce the effects of glaring on any puzzle piece so that the features are more easily identified within the piece itself.

3) Feature Extraction

Once we have the piece we are looking to match isolated, we can run our feature extraction method to try to find parts of the puzzle piece with notable features like corners and edges, and map them to that piece.

4) Feature Matching

Once we have these features, we are able to match them with the features extracted from the puzzle's final image. From there we will perform different confidence checks to ensure that the algorithm is confident that it has found the right piece. These features are also fairly rotationally robust, meaning when the user taps a piece that piece will not have to be in the correct orientation for it to be recognized as the correct piece by our algorithm.

Once the system is confident, it will communicate this information to the software back-end, which will display a graphic showing a circle around the piece, a circle around where that piece should go, and an arrow pointing between them. These graphics are discussed further in Section V.D.

C. *Tap Detection*

The Tap Detection & Localization component will run as an independent Python 2.7 process and will interact with the Leap Motion controller over USB 2.0 and the software back-end via local socket. The Leap Motion v2 SDK will be used to interact with the controller, which provides functions for examining frames and identifying tap gestures. The service will communicate with backend using a very simple API. The back-end triggers the service by sending a single byte of data: a "T" character. The service then waits until it detects a tap, identifies the coordinates of the tap in its own 3D coordinate system, makes sure that the tap was within the bounds of the work surface (in 3D space, so taps in air or outside the frame will be filtered), and then sends the 2D coordinates back to the back-end as 2 floats, using Python's built in functions *struct.pack* and *struct,unpack* to make sure the data is as small as possible. There will be no time-out period.

D. *Animations & Display Builder*

The Animations & Display Builder component will be a Python 3.8.1 library that uses PyGame as a graphics library. It will output the PyGame display via HDMI to the projector and provide functions for the back-end to call to display animations and text to the user. There are only a few unique animations and graphics necessary for our service, limited to: a general text display, animated circles of various sizes for identifying locations, and arrows for tracing paths between locations. Thus, the library will provide 6 functions for adding and removing these graphics from the outputted display, as well as initialization and teardown functions for the display.

E. *Software Back-end*

The software back-end will be a Python 3.8.1 process that orchestrates the operations of the three preceding software components and runs the user interaction cycle, as well as initialization of the puzzle solving. It will provide the prompts to the Display Builder to instruct the user, trigger the CV system to store the solved puzzle solution when it is placed below the webcam by the user, and then enter the puzzle solving cycle. The cycle is composed of a few steps. First, the back-end adds a prompt instructing the user to tap a piece to the display and triggers the tap detection service. The user taps a piece, and when it is reported by the tap detection service, the back-end removes the text prompt, adds a circle animation around the coordinates of the tap, and reports the coordinates to the CV library. Once the CV library responds with the coordinates of the identified piece and the suggested location, the back-end clears the display and adds animations for both locations and an arrow between them. Finally, after some time, the back-end clears the display and calls the CV system to check if a completed rectangle could be identified, signaling the end of the puzzle solving loop. If one cannot be found, the puzzle is still incomplete and the back-end outputs the text prompt and starts the loop again.

A key component of the back-ends operations is re-mapping coordinates in between systems, since each device will have a different point-of-view and coordinate system. Since we will know the final positions of all the physical components, we will hard-code this transformation once we have measured and calculated it once. This allows each device to report native coordinates and allows the back-end to perform whatever conversion are necessary between each system.

VI. Project Management

A. *Schedule*

Our schedule is expanded at the end of the document. In general, each team member has a task to complete every week. In general, we've been following a cycle of research, then into implementation-refinement cycle, and then into integration. This has worked well so far, but we have yet to integrate the pieces together.

B. *Team Member Responsibilities*

Andrew is taking the lead on the Computer Vision aspects with a secondary responsibility in the integration of the software with the raspberry pi and hardware integration insofar as it is helpful for the computer vision. Aneek is taking the lead on gesture controls and shares a bulk of the integration of the project with Connor. Connor is heading the projector interfacing and design of the frame, as well as helping Andrew with the Computer Vision aspects.

C. Budget

We have utilized less than half our budget so far. The complete Bill of Materials and their associated costs is included at the end of the report.

D. Risk Management

So far, our risk management has revolved around alternative computational methods, such as exchanging the raspberry pi for a laptop or AWS, and giving ourselves some slack. As showcased in the budget subheading, we have plenty of budget left in case we need to pivot our design or if we break something or there are any other surprise budgetary needs, we have left room for those extra pieces as well as potentially rushed shipping. We also have given ourselves the ability to forego the Leap Motion and go back with our fall back design which would involve little more computer vision, where we designate a spot on the workspace for the user to place a piece for which then the computer vision pipeline is run.

VII.RELATED WORK

There is very little related work available commercially. We have seen some other attempts on physical puzzles, but they were only preliminary results and didn't end up working. We also found a simulation that was able to match pieces to a puzzle, but they had the pieces in the right orientation originally. Furthermore, the simulation was only working with virtual pieces which used perfect pieces, instead of the imperfections we are dealing with by using real world pieces.

VIII.SUMMARY

We are hoping that our project will be able to meet our requirements, but since we have not fully completed our project, there could be bugs we run into.

A. Future work

This will depend on how far we get, but we would like to expand this algorithm to not require a visual of the box front. This felt like it would be out of the scope of the time and budget constraints.

B. Lessons Learned

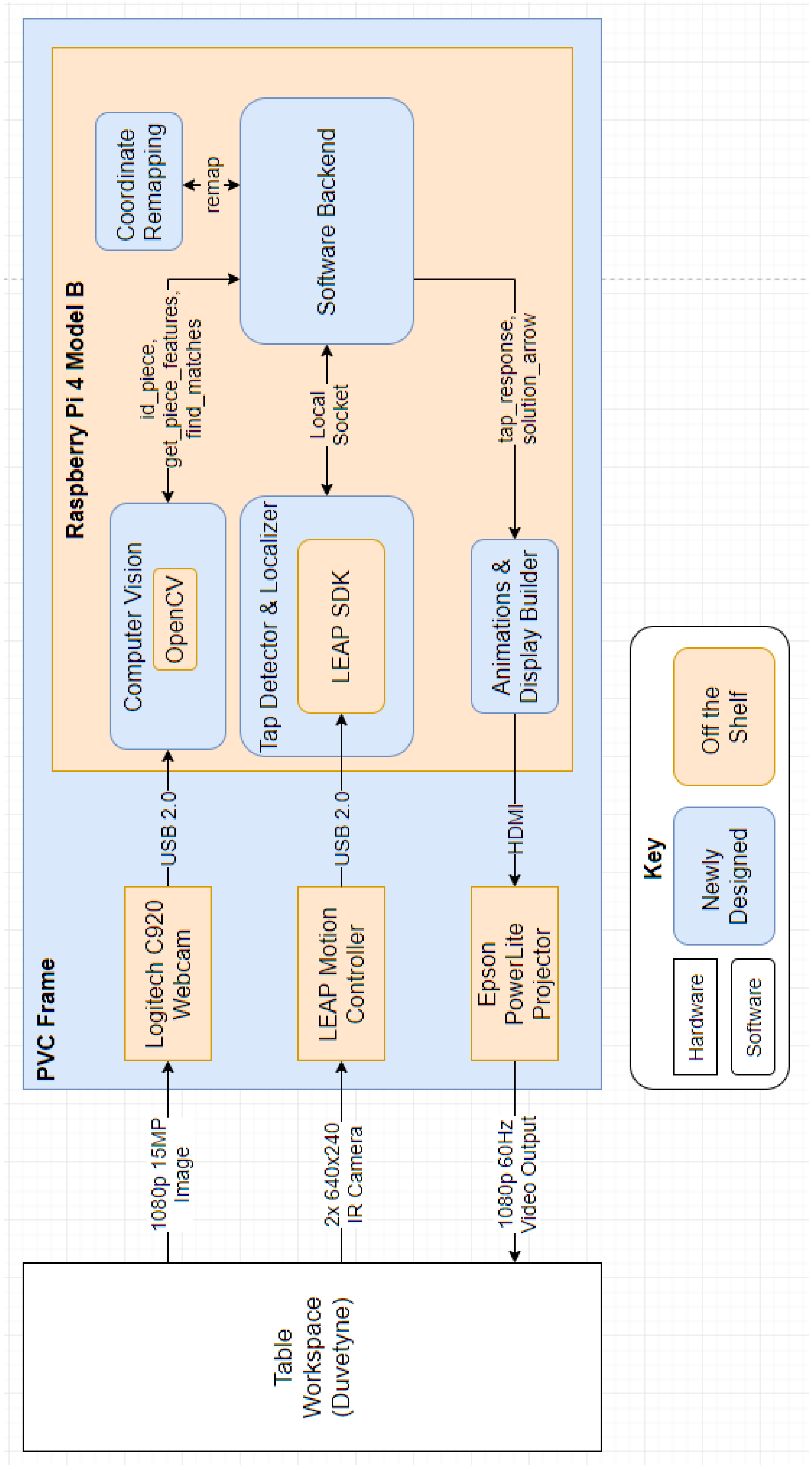
We think that there is a lot that can be learned from accurately and realistically giving yourself a good schedule. We think that a lot can be learned by accurately tuning your values to grab the relevant images as well as making sure that you take into account your development platform early on into development.

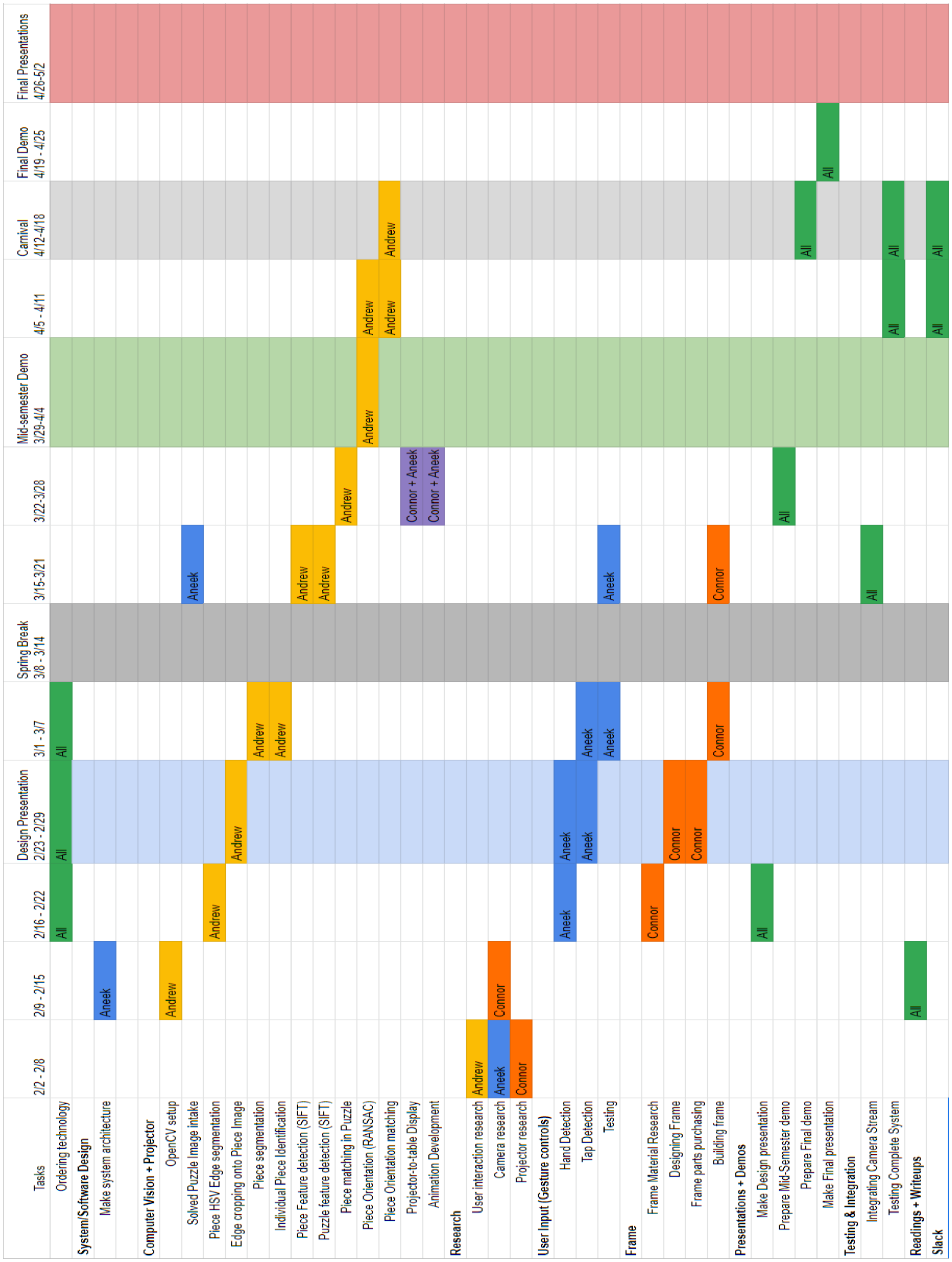
REFERENCES

[1] Leap Motion Datasheet, https://www.ultraleap.com/datasheets/Leap_Motion_Controller_Datasheet.pdf

[2] OpenCV, https://opencv.org/

[3] Research paper on Leap Motion latency, https://pdfs.semanticscholar.org/3aab/d55945b1460620e78ff040e23a819f1523dc.pdf


PVC Frame
Raspberry Pi 4 Model B
Coordinate Remapping
remap
Software Backend
id_piece, get_piece_features, find_matches
Local Socket
tap_response, solution_arrow
Computer Vision
OpenCV
Tap Detector & Localizer
LEAP SDK
Animations & Display Builder
USB 2.0
USB 2.0
HDMI
Logitech C920 Webcam
LEAP Motion Controller
Epson PowerLite Projector
1080p 15MP Image
2x 640x240 IR Camera
1080p 60Hz Video Output
Table Workspace (Duvetyne)
Key
Hardware
Software
Newly Designed
Off the Shelf



Tasks	2/2 - 2/8	2/9 - 2/15	2/16 - 2/22	Design Presentation 2/23 - 2/29	3/1 - 3/7	Spring Break 3/8 - 3/14	3/15-3/21	3/22-3/28	Mid-semester Demo 3/29-4/4	4/5 - 4/11	Carnival 4/12-4/18	Final Demo 4/19 - 4/25	Final Presentations 4/26-5/2
Ordering technology			All	All	All								
System/Software Design													
Make system architecture		Aneek											
Computer Vision + Projector													
OpenCV setup		Andrew											
Solved Puzzle Image intake							Aneek						
Piece HSV Edge segmentation			Andrew										
Edge cropping onto Piece Image				Andrew									
Piece segmentation					Andrew								
Individual Piece Identification					Andrew								
Piece Feature detection (SIFT)							Andrew						
Puzzle feature detection (SIFT)							Andrew						
Piece matching in Puzzle								Andrew					
Piece Orientation (RANSAC)									Andrew	Andrew			
Piece Orientation matching										Andrew	Andrew		
Projector-to-table Display								Connor + Aneek					
Animation Development								Connor + Aneek					
Research													
User interaction research	Andrew												
Camera research	Aneek	Connor											
Projector research	Connor												
User Input (Gesture controls)													
Hand Detection			Aneek	Aneek									
Tap Detection				Aneek	Aneek								
Testing					Aneek		Aneek						
Frame													
Frame Material Research			Connor										
Designing Frame				Connor									
Frame parts purchasing				Connor									
Building frame					Connor		Connor						
Presentations + Demos													
Make Design presentation			All										
Prepare Mid-Semester demo								All					
Prepare Final demo											All		
Make Final presentation												All	
Testing & Integration													
Integrating Camera Stream							All						
Testing Complete System										All	All		
Readings + Writeups		All											
Slack										All	All		

Item	Cost	Description	Status
Epson Powerlite 1776W Projector	0.00	Borrowed from ECE dept.	Arrived
Logitech C920 Webcam	69.99	Purchased on Amazon	Waiting
Raspberry Pi 4 Model B	73.11	Purchased on Adafruit	Arrived
Duvetyne Sheet	22.68	Purchased via independent distributor	Waiting
Leap Motion controller	106.64	Purchased on Adafruit	Waiting
PVC piping	TBD	PVC pipes are not yet purchased	Yet to be purchased
Total	272.42		