Portfolio Index

Dynamic Keyboard Layers

Skills Used

- *AutoHotkey*
- *Software Development*

Rover Concept Pitt Rocketry

Skills Used

- *3D Modeling*
- *Top-Down Design*
- *3D Printing*

Scouting Data Dashboard

Skills Used

- *Python*
- *Software Development*
- *Data Analysis*
- *Communication/Collaboration*

SOL(Robotics)

Skills Used

- *Robotics*
- *3D Modeling*
- *Top-Down Design*
- *Sensor Design*
- *Java Programing*
- *Machining*

Dynamic Keyboard Layers

Overview

This is an implementation of dynamic keyboard layers. This is a complex concept to explain but it quickly becomes second nature to use.

I first need to explain what a normal keyboard layer is. A keyboard layer is a set of key bindings that can be reached by holding a modifier key. The most widely used example of this is the shift key. Normally keys are mapped to lowercase letters. When the shift key is held the “uppercase layer” is activated AND THE KEYS BECOME MAPPED TO THEIR UPPERCASE EQUIVALENTS. This is exactly how the script works. By holding a toggle key you activate a dynamic layer and the functionality of your keys change.

The dynamic aspect of these keyboard layers comes into play because the keyboard layer that is activated is dependent on the application a user is currently using. This means that the “6” key on this layer could be used to change tabs when a user is using a browser and select a line when they are using their favorite text editor (VSCode in my case). This happens with no input from the user to switch the key binding for the application being used.

I use this script along with my tiling manager daily and it makes interacting with my computer infinitely more intuitive.

Configuration

Each keyboard layer along with the toggle key are completely configurable through a configuration utility. This utility writes each layer to a config file in a folder called “Hotkeys” in the same directory as the script itself. These files include: the application the layer applies to, keys to send on a key down event, and keys to send on a key up event. There are also two special configs.

The first special config defines the keys to include in the keyboard layer(s) as well as what should be sent when those keys are in their untoggled state. I use this apply the dynamic layers to F13-F24 so I don’t remap any of the keys normally on my keyboard, these keys send 0-9, “-“, and “backspace” normally. These mappings make typing numbers with just one had simple, it’s basically a numpad.

The second special config is the “default” config. This is the layer that is activated in applications that don’t have their own key map. I have this mapped to be common hotkeys like ctrl-s, and alt-tab.

Hardware implementation

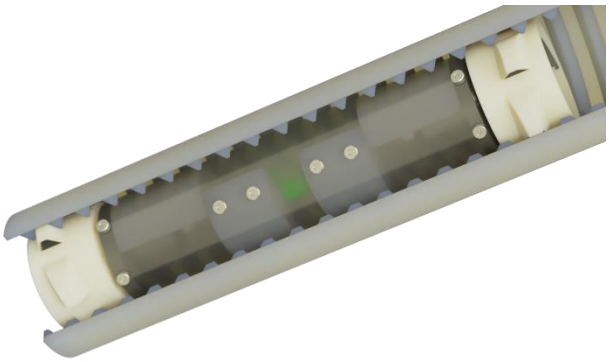
I’m currently working on implementing all the functionality of my script on a mouse. This would let users bring the functionality with them wherever they go. I’ve been working with my roommates to make this a reality. We’ve gone to a few pitch competitions in order to get initial funding so we can build our first hardware-based prototype. Our long term goal is to sell the mouse as a full product. I wrote all the code in our prototype for my own personal use I’m working with my roommates now to mimic this functionality in the hardware on a mouse.

I’ve been developing, maintaining, and using this script for about 5 years. It’s all written in AHK a scripting language for windows.

Click or scan the QR code to see a GIF that demonstrated the dynamic layers. You can also find a link to the code on my GitHub.

Rover Concept Pitt Rocketry

Overview



This rover was designed as a part of Pitt's 2019-2020 USLI payload. It was tasked with surviving a rocket flight then collecting an ice simulant made of Injection molding pellets.

The Idea behind this rover was that with just two motors and some fancy geometry we could create a rover capable of retaining itself inside a rocket during flight, deploying from that rocket after landing, then traveling across a field and collecting the ice simulant.

Wheels

That Fancy geometry came in the form of these wheels with thread shaped scoops. The threads allow the drone to screw into the rocket body keeping the rover secure during flight. These wheels can then spin relative to the rocket body as the main body of the rover is keyed to the thread preventing it from spinning. To test this a wheel and small portion of thread was 3D printed. It was successful in testing so this design has lots of merit.

The scoop portion of the wheel was based of our team's previous design that successfully used similar scoops/flaps to collect the same simulated ice pellets in the previous year's NASA USLI competition.

Implementation

We didn't get past the prototype phase on this design. We instead decided to go with a drone so we could pursue vision processing for fully autonomous collection. Even though I didn't get the chance to fully realize this design, I'm still proud of the CAD work that went into it. I gained lots of experience with surface modeling designing the wheels. I also go the chance to simulate the forces on my design using past flight data which was a new experience.

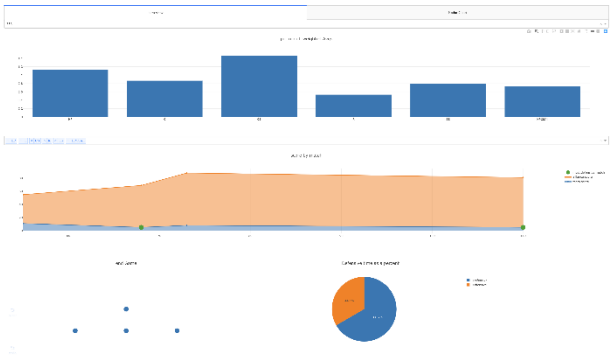
Click or scan the QR code to view the CAD for this design in its entirety on my website.

Scouting Data Dashboard

Overview

This is a data dashboard I built for my robotics team using Plotly. It was my first python project, and even though I've gotten significantly better at writing python code, I'm still proud of the functionality. I built this dashboard to quickly gain insights into a team's performance before a match. Before this dashboard, that insight would come from the eye test alone, which meant I would need to watch at least one match from every team before they became one of our two randomly assigned alliance partners.

Main Screen



The main screen of the dashboard had data about what a team did during match play. With a glance at this dashboard, I could see each team's strengths and begin to strategize around those.

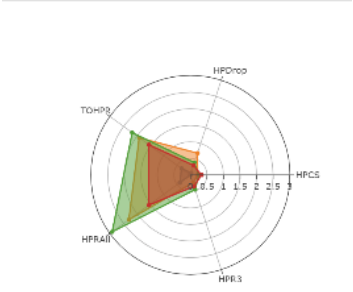
The bar chart on the top has its own dropdown where you can select from a list of all the scoring categories. This can give you an idea of how a team performs on average.

The second graph shows you a team's score by match in blue, and their alliance's score in orange. The team's score is calculated based on scouting data our team collects. This data is pulled from a google sheet automatically. The alliance's score is then pulled from the Blue Alliance API. The Blue Alliance has the official scores for every match, so it's an invaluable resource when analyzing a team's performance. There's a green dot on the match(es) where the team plays the most defense on this same graph. This dot lets us go back and watch film from that match to see how a team defends and drives.

Finally, there's a chart showing what level(s) a team has gotten to in endgame, as well as a diagram showing how much a team plays defense.

Combining these four graphs gives us an idea of how a team will play before even looking at their robot or talking to their drivers.

Radar Chart



This radar chart was built using the same data behind the main screen. It was created specifically for alliance selection. Alliance selection happens before the playoffs of an FRC competition. The top 8 teams in the event chose their alliance partners before the playoffs in a snake draft. I chose a radar chart to display this data because by looking at the shape that a combination of teams made, you could quickly build a well-rounded alliance. This didn't work very well in practice, so we mostly relied on the main screen for data analysis.

Click or scan the QR code to view the scouting data dashboard on my website.

Sol

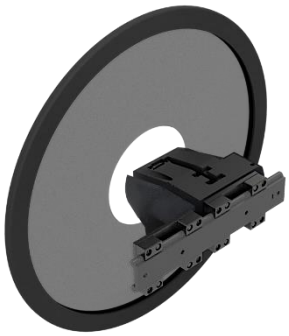
Overview

Sol was the 20th anniversary robot of FRC team 333. It was also the robot I helped design and manufacture in my senior year of high school. Sol competed in two regional events as well as the Tesla division in the championships in Detroit. It did so without any significant failure. The only thing we had to replace was a limit switch. Sol was designed with top-down design principles in Fusion 360. This was to help facilitate collaboration in both CAD and CAM. This was the third robot I worked to build on Team 333.

Machining

One of my main contributions to this robot as the CAD/CAM lead was in machining the parts for this robot. I created and ran toolpaths for our Tormach CNC mill and X-Carve CNC router to make the robot's parts. To accomplish this, I created toolpaths in Fusion 360, brainstormed and implemented workholding to keep the raw stock secure during machining, probed stock, established work coordinate system origins, and created/maintained tool libraries for both machines. Being the one making the parts I designed was an invaluable experience. Manufacturing these parts gave me insight into what makes a part manufacturable.

Hatch Panel Mechanism



The hatch panel mechanism was designed to slide and pivot any way it needed too in order to passively grab a hatch panel from the human player station. The mechanism has an embedded sensor that uses two LED's and a photosensitive resistor to detect when it has hatch panels in it's possession. When it comes time to let the hatch panel go this mechanism uses a single-acting pneumatic cylinder to release and score the hatch panel. I did the design work for this mechanism and it's embedded sensor.

Click or scan the QR code to view the CAD for this design in its entirety on my website.