Tecnológico de Monterrey

Mexico City Campus

Implementation of Mechatronic Systems (MR2004B)

Integrative report

Students:

Leonardo Valero Perales	A01658516
Frida Sophia Chávez Juárez	A01665457
Maximiliano Elizarraras Shchelkunova	A01666610
Victor Manuel Galicia Hernández	A01666471
Oscar Gadiel Ramo Martínez	A01665807

Professors:

Rodrigo Regalado García

Ricardo Enrique Gutiérrez Carvajal

Artemio Urbina García

Due date: June 14, 2025

INTRODUCTION

Automation empowers. That is the motto that Tec de Monterrey displays on their web page for Mechatronics Engineering, a motto that becomes truer by the day. One area where automation opens many possibilities is that of autonomous vehicles: whether it's self-driving cars, automatic pilots for passenger aircrafts and trains, or surveillance drones, automating processes and objects promises to lower costs, increase efficiency, and reduce risk.

Autonomous Guided Vehicles, also known as AGVs, are robots that follow a path using an array of sensors ranging from line followers, to full computer vision systems like LiDAR, in order to fulfil a specific task, usually transporting something within an industrial context.

With the objective of furthering our capabilities as mechatronics engineers a challenge was presented to us: design and manufacture an AGV with a line follower that led it down a path with an objective marked at the end. Along the trip it would couple with a scissor lift carrying a predetermined amount of beans. After the coupling, a forward collision avoidance system had to kick in and stop the vehicles when presented with an obstacle in its tracks. After reaching the endpoint, the material had to be dumped from the scissor lift by means of the carrying basket rising and then tilting.

THEORETICAL FRAMEWORK

1. Automated Guided Vehicles (AGVs)

   Definition and Working Principle
   Automated Guided Vehicles are mobile robots used in industrial settings to transport materials without human intervention. AGVs follow predefined paths using guidance technologies such as magnetic strips, laser navigation, vision systems, or LiDAR. They are equipped with sensors and software to manage navigation, obstacle avoidance and task execution.

   Common Types of AGVs:
   - Towing AGVs: Pull trailers with loads
   - Unit Load AGvs: Transport single units like pallets or containers.
   - Forklifts AGVs : Automates versions of forklifts for vertical lifting.
   - Assembly Line AGVs: Integrates into production lines to move products through assembly stages.

   Industrial Applications of AGVs
   - Automotive Industry: AGVs transport parts between stations, reducing labor costs and improving accuracy.
   - Warehousing and Distribution:AGVs handle inventory movement, especially in e-commerce logistic centers.
   - Pharmaceuticals: Used in clean room environments for sterile, precise delivery of medicines and raw materials.
   - Food and Beverage Industry: Transport of packing materials and finished goods with minimal contamination risk.

   Limitations of AGVs

- Fixed routes: Limited flexibility unless advanced navigation systems are used.
- High Initial Cost: Expensive setup and infrastructure modifications.
- Limited Load Capacity: Cannot replace heavy-duty transport equipment.
- Complex Maintenance: Requires skilled personnel for troubleshooting and updates.

2. Scissor Lifts

Definition and Working Principle
A scissor lift is a type of mobile elevating work platform (MEWP) that raises personnel or materials vertically using a system of crisscrossing supports that elongate when hydraulic, pneumatic, or mechanical power is applied. The lift mechanism resembles the action of scissors opening, hence the name.

Common Types of Scissor Lifts:
- Hydraulic Scissor Lifts: Operate using pressurized hydraulic fluid.
- Pneumatic Scissor Lifts: Use air pressure, ideal for environments requiring no oil leakage.
- Electric Scissor Lifts: Battery-powered, suitable for indoor use.
- Diesel Scissor Lifts: Fuel-powered and more suited to rugged outdoor terrain.

Industrial Applications of Scissor Lifts
- Construction Sites: For lifting workers and tools to elevated areas safely.
- Maintenance and Repair: Access to high ceilings, HVAC systems, or lighting in factories and warehouses.
- Manufacturing Assembly Lines: Elevate components to optimal working heights for ergonomic assembly.
- Aviation Industry: Used in aircraft maintenance and servicing operations, especially for reaching aircraft doors and fuselages.

Limitations of Scissor Lifts
- Limited Horizontal Reach: Designed only for vertical lifting.
- Platform Size Constraints: Space on the platform is limited.
- Height Restrictions: Only reaches moderate heights compared to boom lifts.
- Mobility Challenges: Not always easy to maneuver in tight spaces or uneven terrain.

DEVELOPMENT OF PROTOTYPES AND THEIR TECHNOLOGIES

Mechatronics. It is in the discipline's very name. Mechanical and Electronic systems are the bread and butter of each and every project. It doesn't matter how reliable and optimized your

electronics are if they don't have a proper place to go in: one that protects them from external factors and provides them with a solid foundation that allows cables to stay in place, sensors to give reliable readings, and motors to use as a foundation. This is why reliable housings had to be developed for the project's sensors and actuators.

Proximity sensors were needed in order to avoid collisions with oncoming objects. As such, they needed to be at the very front of both the AGV and scissor lift's structures due to the need for clear lines of sight. In order to reach this position we decided to place a protrusion that extended beyond the AGV's body and above the lift's bottom base. The structure was quite simple as it consisted of one single aluminum layer placed in an inverted "L" shape on top of the wall on which the line followers rested. The sheet's height had to be greater than the RHS rods that connected the lift's wheels while being lower than the scissor lift's electronics base. This meant it had to be placed at a height greater than 7 cm but lower than 12cm; a decent margain.

In order to minimize risk of not having enough distance to break, the ultrasonic sensor was placed 7 cm in front of the AGV's front wall. Yet another consideration that had to be taken into account was the need for the aluminum plate to be easily removed as the AGV's electronic circuitry would almost fully be contained within said plate and if anything were to go wrong, easy access would be a must.

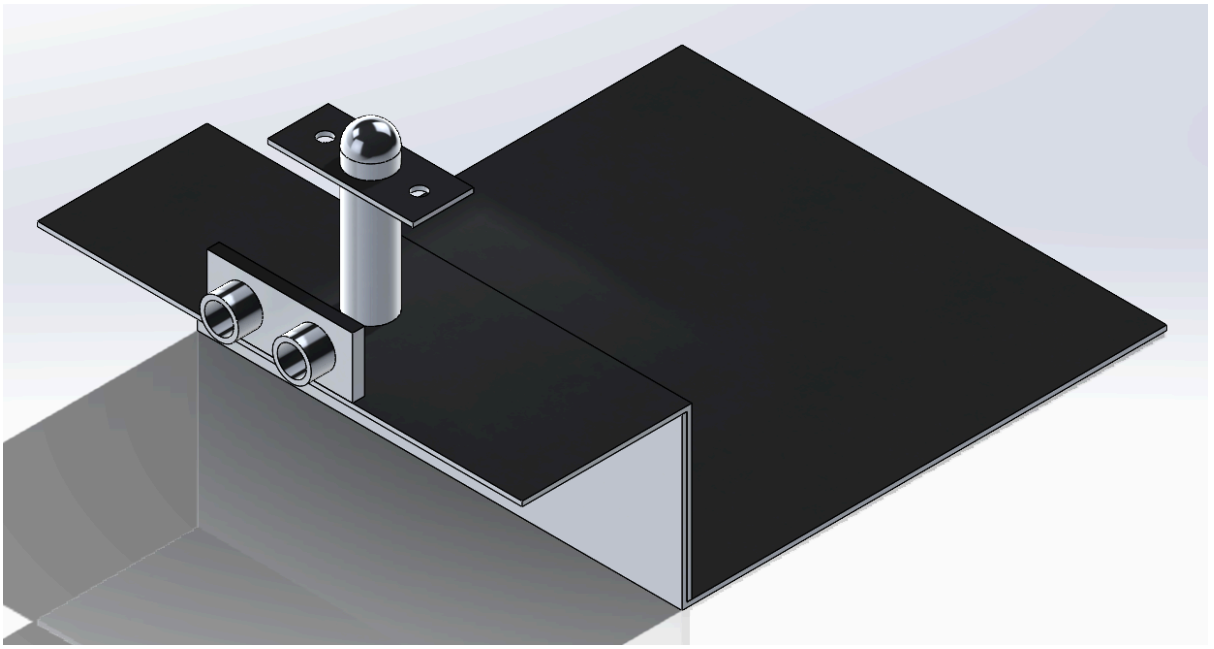

Figure 1: Isometric view of the main AGV plates

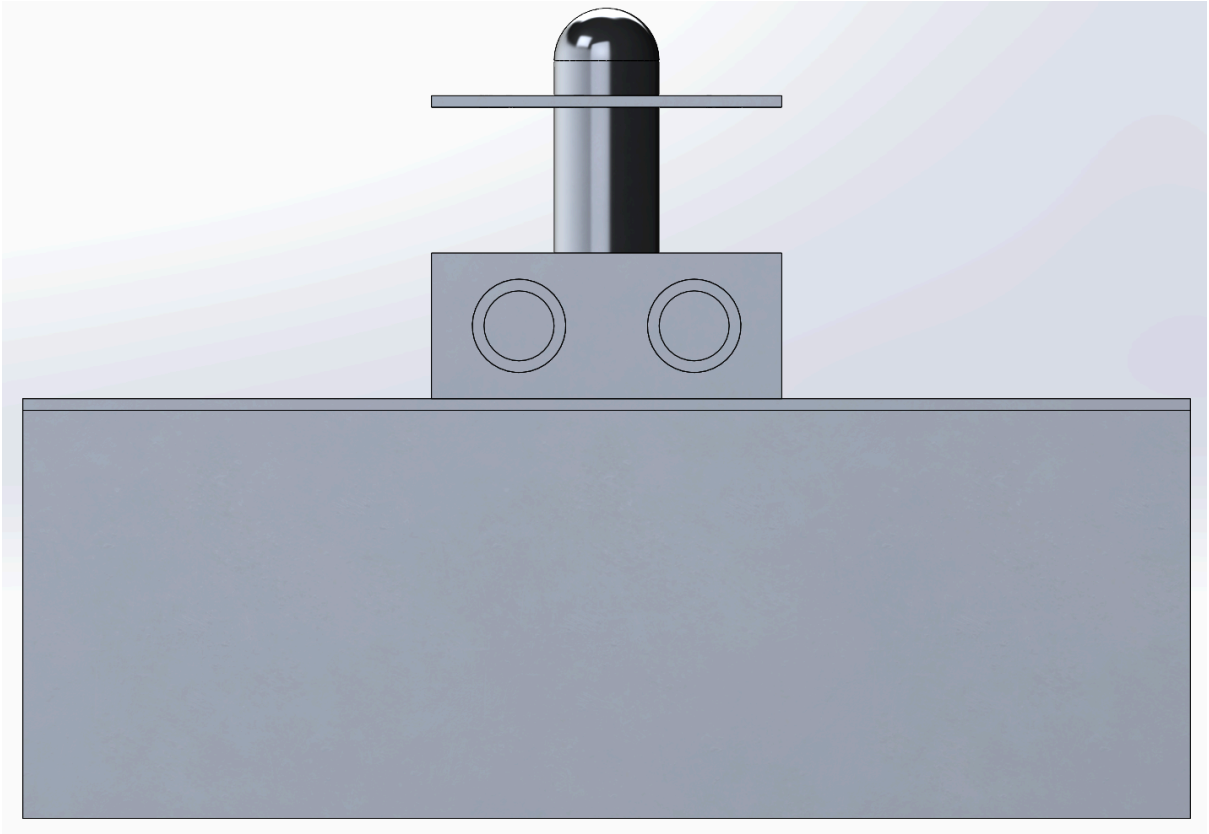

Figure 2: Front view of the AGV's main plates.
The front plate with 2 circles is the ultrasonic sensor.
The "tower" with a sphere on top is the adapted door latch.

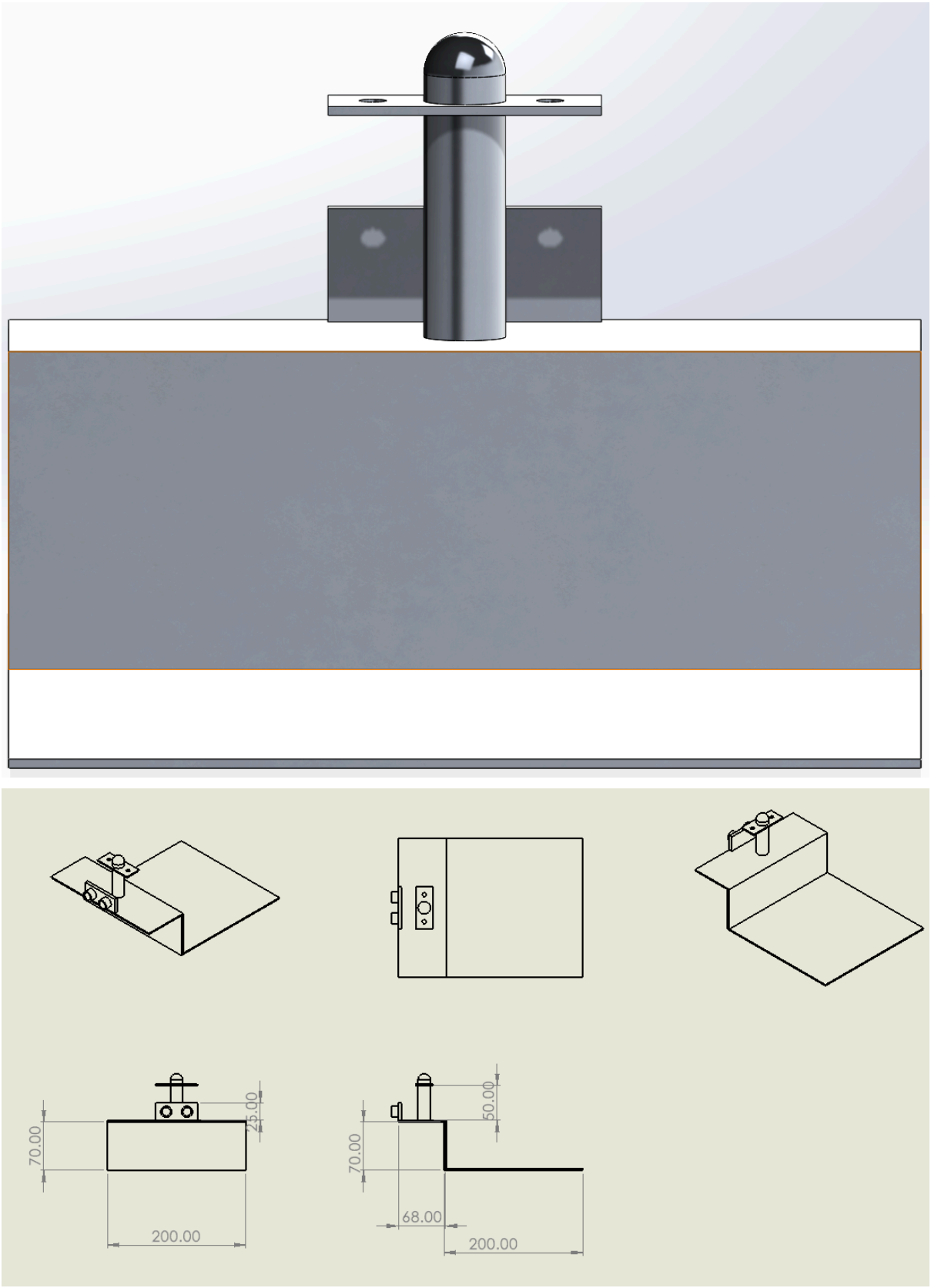

Figure 3: Drawing of the main AGV plates

In order to ensure a reliable connection between AGV and lift a physical interlocking mechanism had to be designed and implemented; otherwise, the risk of the lift not stopping

with the AGV or the 2 of them detaching through transportation would be very much real. To guarantee a tight connection a latching mechanism was scavenged and reused from an old door: this mechanism ensured that the AGV could slide into the lift without any problem, but it couldn't detach on its own after said connection was performed.

This system bypassed the problems that magnets have of sticking to any surface that's ferromagnetic, not just a specific point as door latches do, however, that also comes with risks, mainly the fact that if the latch misses the strike plate and the whole within, it is useless as it will stay contracted and without a connection. To eliminate this issue a guiding mechanism was implemented by placing a pair of angular profiles in a "V" shape on the bottom part of the lift's base that allowed the latch to enter within a wide area but as the AGV moved forward it would be forced into a tighter and tighter spot until it reached the center point where the latch would spring back up again and therefore couple the 2 vehicles together.

The latch was placed 5 cm above the same "L" shaped plate on which the proximity sensor was located in order to almost touch the lift's base plate and prevent a collision but still allow for the latch to couple to the lift.

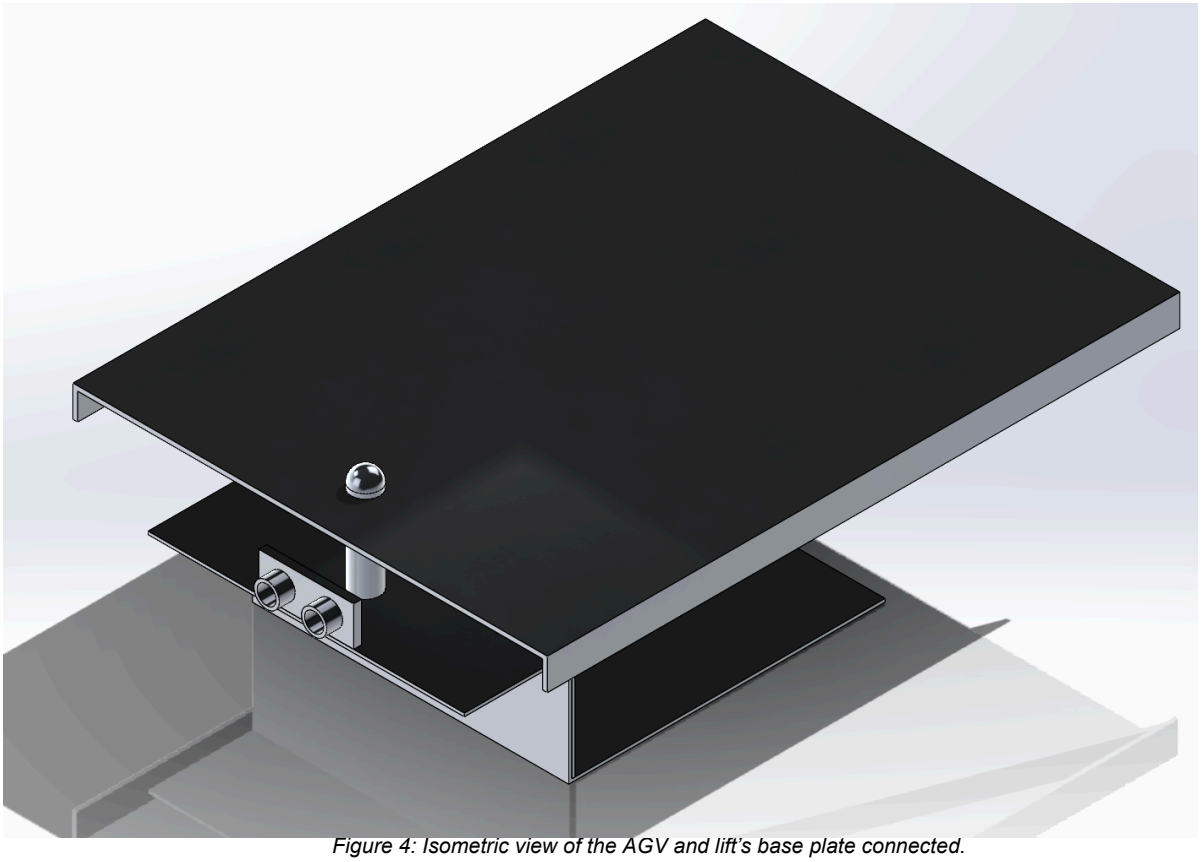

Figure 4: Isometric view of the AGV and lift's base plate connected.

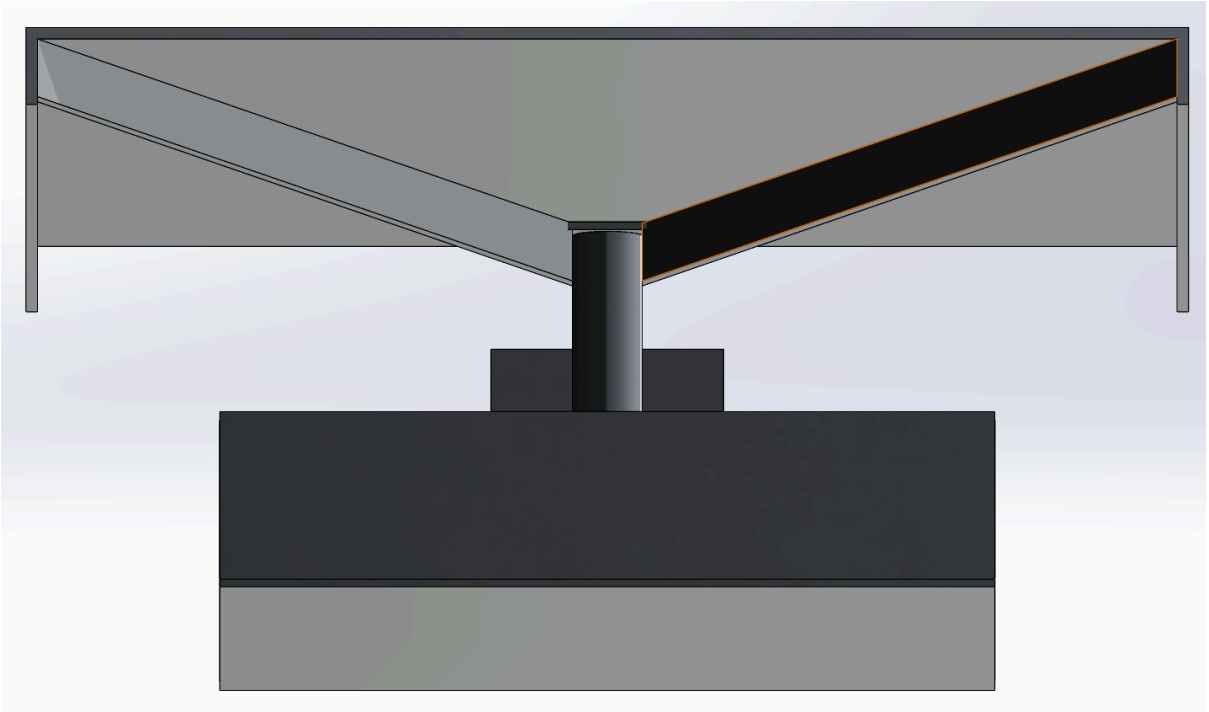

Figure 5: Back view of the AGV and lift's base plate connected.
The guiding system can be easily seen in the form of the "V" shaped structure.

Figure 6: Side view of the AGV and lift's base plate connected.
The latch can be seen protruding from the base plate.

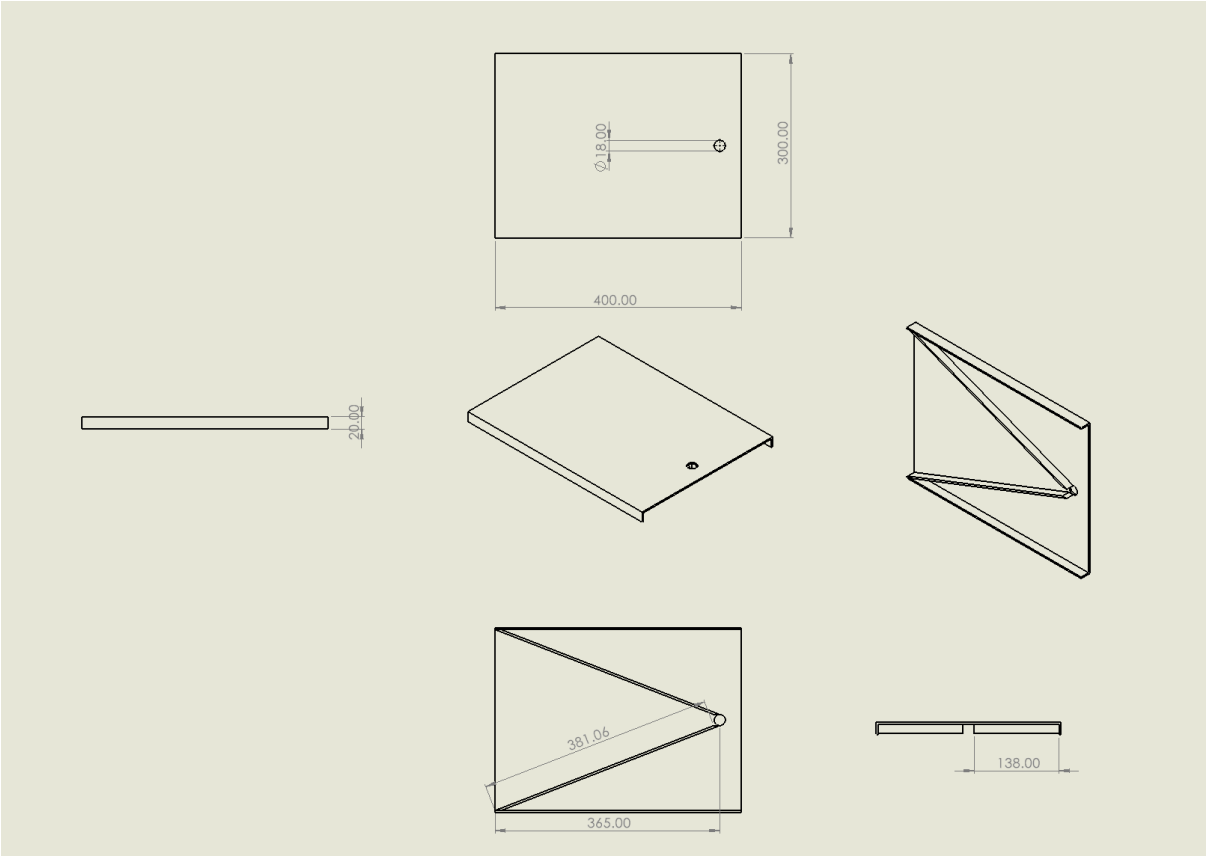

Figure 7: Drawing of the lift's base plate.

Wheels are the cornerstone of any mobile system, and their implementation within the AGV was no simple task. We decided on using a 3-wheel system with the front 2 wheels, made of a plastic rim and rubber wheel, acting as the drivers thanks to their 2 independent motors, and one free-spinning wheel placed at the back of the vehicle. For efficient and reliable motion, the front wheels had to be correctly attached and aligned to their respective motors; a task that proved harder than expected due to the different measurements of the motor's shaft compared to the wheels' center bore and the lack of a direct way to transmit the torque as the motors had no way to spin the wheels other than the friction between the motor's shaft and wheels' center bore which caused the motors to continuously freespin without moving the wheels.

To solve both of these problems we had to first bore out a bigger hole within the wheels' center so that the motor's shaft could fit within. Then we used epoxy glue to ensure a secure connection between the 2 elements and prevent the motors from free spinning. After this we realized that due to the AGV's lightweightness sometimes the rubber wheels would not spin while the inner rims would. To solve this problem we had to add weight to the motors so they would "stick" more to the ground, and we had to use hot silicone to adhere the wheels to the rims.
Finally, the 2 motors had to be correctly aligned to ensure a stable drive and they had to be tightly adhered to the AGV's base. This was accomplished through the use of epoxy glue and straps.

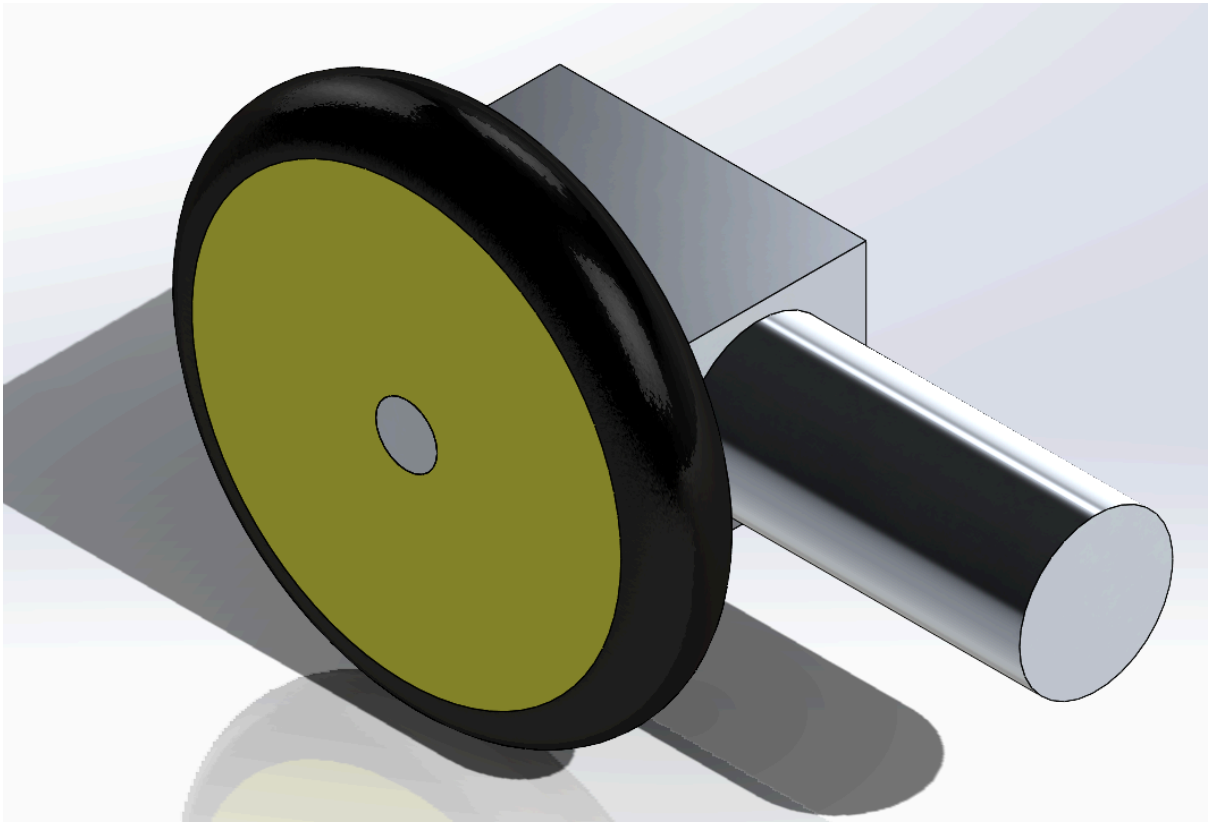

Figure 8: Isometric view of one of the AGV's driving wheels and its respective motor.

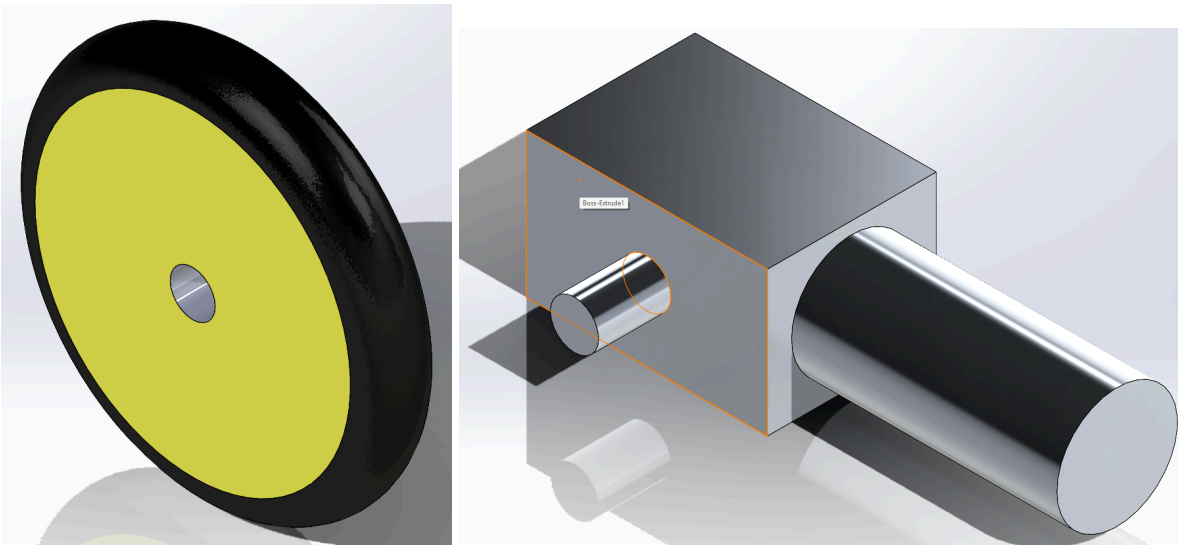

Figure 9: Left: A singular wheel like the ones used in the AGV. The yellow rim and the black rubber wheel had to be adhered together with hot silicon. The rim's hole had to be bored open more than it already was.
Right: A recreation of the motors used to move the AGV and therefore push the scissor lift. The rotor was placed within the bigger cylinder while the gearbox was placed within the box. The rotating shaft is the smaller cylinder.

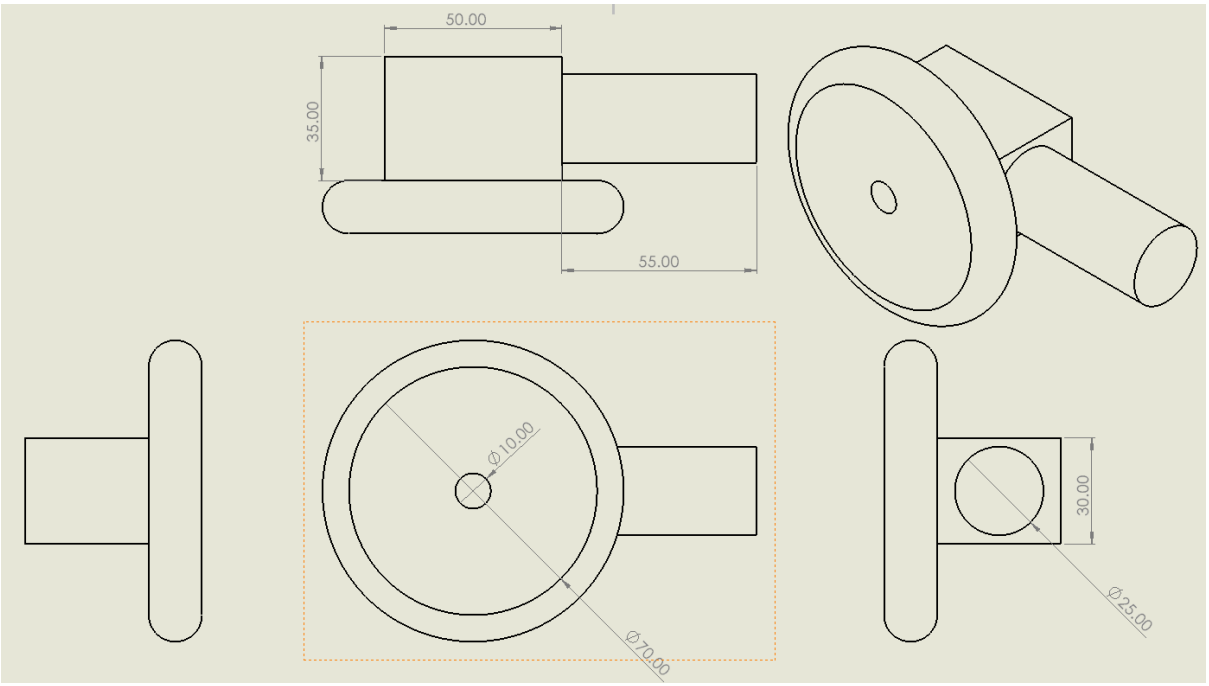

Figure 10: Drawing of the wheel and motor assembly.

Our mechatronic system's ultimate purpose was to deliver a load of beans by transporting them within a solid basket that would then tilt in order to roll them out, therefore, said carrying basket was one of the system's most important components.

To guarantee a reliable point for pivoting, 2 door hinges were bolted to the base of the top of the lift and to the base of the tilting basket. The tilting action was performed by a rack and pinion mechanism placed at the opposite end of the hinges. Once the basket was tilted to an angle of ~30° a door on the same side as the tilting hinges would fall open using another hinge, allowing the beans to fall to their final objective.

However, the amount of beans loaded onto the basket had to be measured by weight. To this end an HX711 load cell was used. This type of load cell must be precisely placed at the basket's center of mass and given ~0.5cm of clearance above and below in order to have space to deform and measure the weight placed upon them. This was accomplished by manufacturing a double-bottom for the carrying basket unto which one end of the load cell would be bolted to, with the other end being bolted to the bottom plate.

No CAD model is available for this section as it had to be improvised due to a misunderstanding on how the requirements for the load cell to work correctly. However, rubric states only 3 designs must have CAD pictures.

MACHINE ELEMENTS

To transmit power and motion to the wheels that drove the AGV forwards we needed a pair of motors that had high torque; speed wasn't nearly as important. That is when we landed on using a pair of JSX634-ZWL-31ZY DC worm gear motors which at 12V have an output rated

torque of 20 kg/cm and a speed of 50 RPM which should be more than enough when taking into consideration the 2 of them that were used.

However, it is known that most motors don't have the same exact speed and torque in their output as they did in their "original" rotor (the part of the motor that is forced to spin by a plethora of magnetic coils). It is thanks to gears that one can control the final speed and torque output: if a pair of gears have a ratio of 1:2, that means that for every full rotation the first gear makes, the second gear will have made 2; this is a **speed** increasing setup. Meanwhile, if a pair of gears have a ratio of 1:0.5 then that means that we have a **torque** increasing setup as for every revolution of the first gear, the second will only make half of one, but with double the torque.

We can compound this effect by adding multiple interconnected gears with different ratios. To save space we can employ compound gears: gears that are of different sizes but, since they share the same shaft, have the same torque and rpm properties. In order to analyse the ratios present within our motors we decided to disassemble them and we found the following: In total, our motors had 4 gears: the first 2 had the same measurements (1:1 ratio) but they were in a worm and worm wheel configuration. After that, the second gear had 30 teeth, while the third gear had 60 (0.5 ratio). The fourth gear also had 60 teeth, but since the previous gear had a compound gear with 30 teeth we again had a ratio of 0.5. Finally, the fifth gear had 75 teeth while the compound gear for the fourth had 30, giving a ratio of 0.4. This means that the final ratio for the whole system was $0.5 \times 0.5 \times 0.4 = 0.1$.

In other words, the original speed at which the motor's rotor spins is 10 times faster $(Motor\ speed = \frac{50\ RPM}{0.1} = 500\ RPM)$ while the original torque was only a tenth of the final output $(Motor\ torque = 0.1 \times 20kg = 2kg)$.

Since our motor's shaft and axle was directly connected to its respective wheel, that ratio stays the same for the absolute last part of the system, but if we had additional external gears or a chain transmission that number could easily change.

Speaking of transmissions, part of our project also consisted in designing a transmission ourselves and since the mobility part of the AGV was complete without additional attachments, we decided to include the transmission in our tilting platform mechanism, which can be visualised in the following manner:

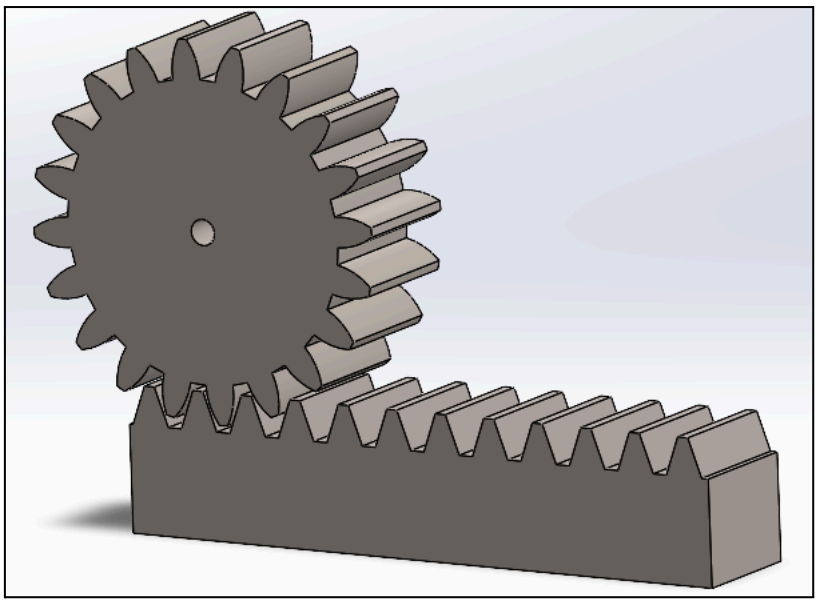

Figure 11. Pinion Rack Mechanism alone

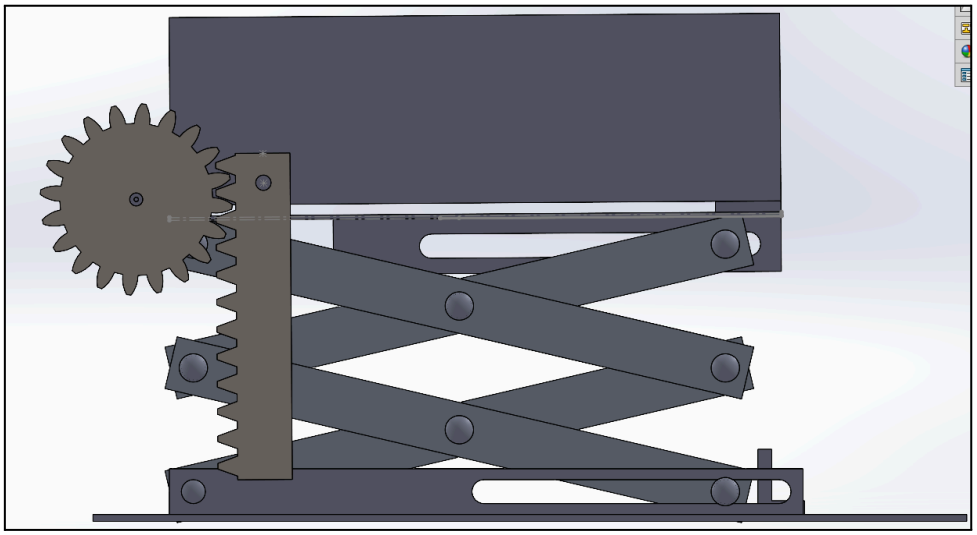

Figure 12. Pinion Rack Mechanism in Scissor Lift.

The pinion is attached to the motor shaft with a *key* while the crank is attached directly to the box with just a screw. That way, when the motor rotates, it causes the rack with the box to elevate.

Despite being quite simple to both manufacture (in 3D printing) and calculate, we thought of it as a way better alternative to a Cam elevation, since less torque was required. That is thanks to its positioning at the opposite edge of the box. And talking about torque, we of course needed to calculate the amount of force our motor required. That was done with a MATLAB code since we did not know how many times the calculations would need to be repeated. That code can be found in the Annex#.

Ultimately, it was obtained that for approximately **6.5 kilograms** of mass (including the box and some more for security measures) we needed a motor with at least **18.3 kg/cm**. That motor ended up being: Stepper Motor Nema 23 of 12V, 2.8 A.

MICROCONTROLLERS

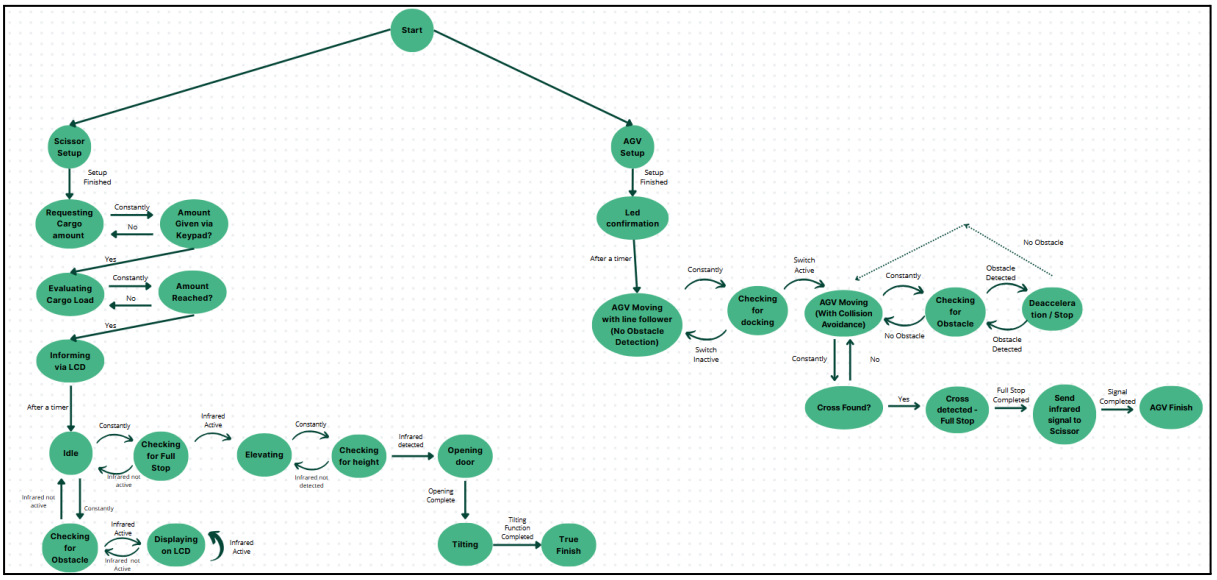

Figure 13: General State Flow Chart

That would be the general state flow chart, however we consider it important to give additional smaller flow charts to setups that are somewhat lengthy and have more options of being active than (on/off). For example, the LCD and Keypad components.

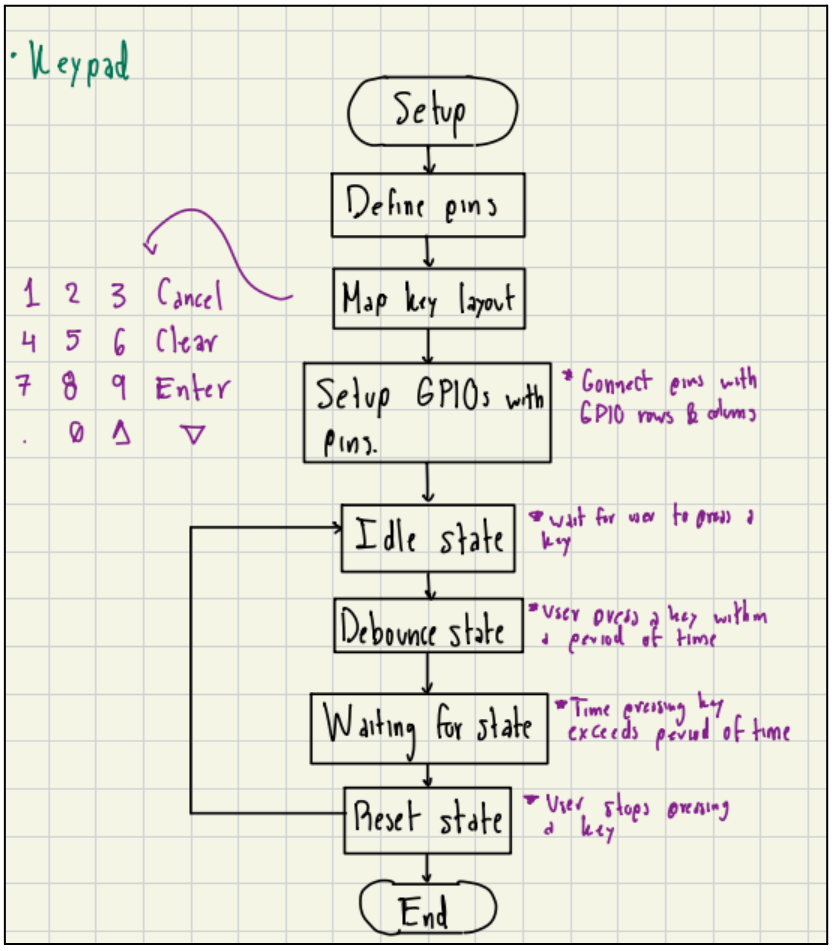

Figure 14: Keypad Individual States

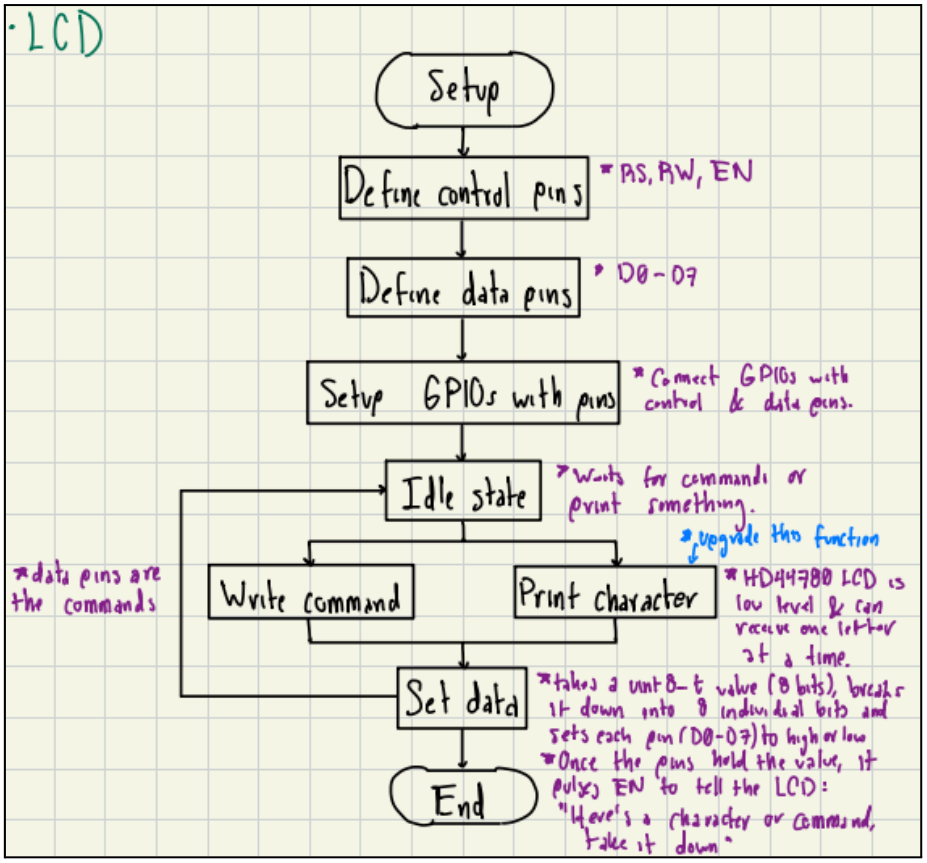

Figure 15: LCD Individual Generic States

ELECTRONICS

The whole system was controlled by two ESP32 Wroom32d, one for the AGV and one for the Scissor Lift. The system worked with 2 12V batteries of 4.5 Ah with a step down 12V to 3.3V module to feed the ESP32.

Figure 16. Battery and Step Down Module

AGV

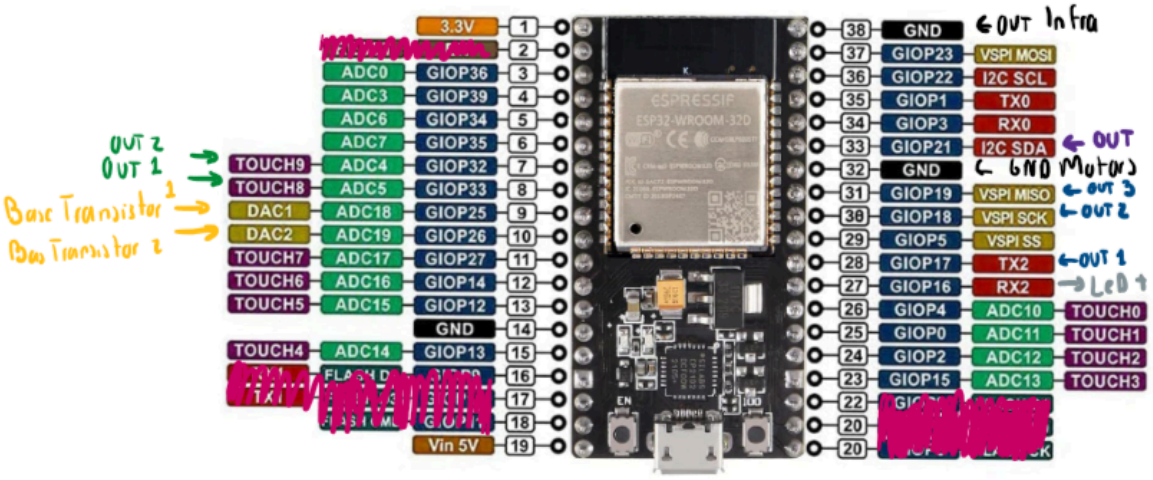

Figure 17. Connections tof ESP32 for AGV

The systems that conform the AGV are:

1. Control for wheels

   The wheels are controlled by two DC gear motors of high torque:

Figure 18. Gear Motor for Wheels

As we know, the ESP32 cannot give a voltage of 12 V and a DC motor does not have a signal input, meaning that we had to find a way for the motors to know when to start and when to stop. To achieve this we used a circuit with a TIP120 transistor that acts like a switch and a 1N5822 diode that acts like a protection system for the ESP.

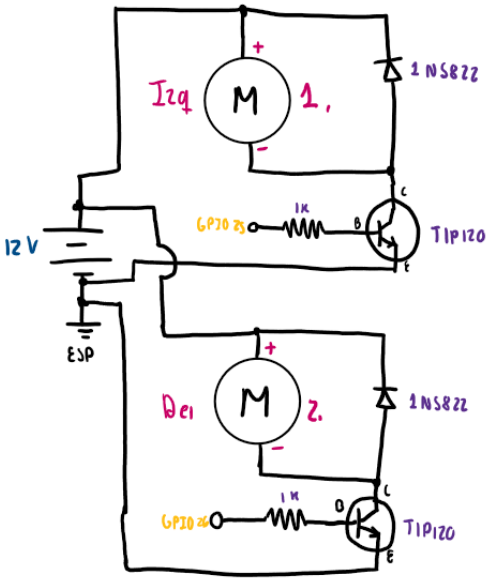

Figure 19. Schematic of circuit for Wheels

2. Line follower
   For the line follower we used 2 infrared sensors that sent the signal to the ESP32 so that the motors could work.

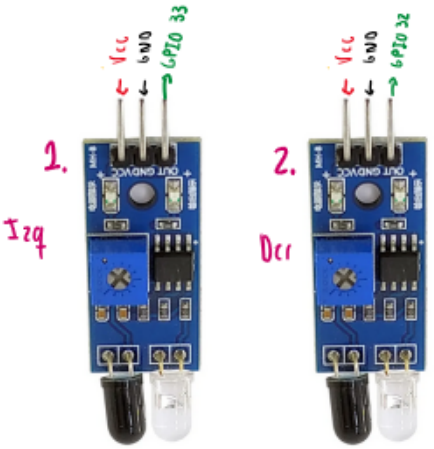

Figure 20. Infrared sensors used as Line Followers

3. Obstacle sensor
   We used an ultrasonic sensor as an obstacle detector, additionally we connected a voltage regulator, from 5V to 3.3V to protect the ESP from the 5V that the sensor provides. Also, a Limit Switch was connected so that the ESP could know when to turn on the obstacle detector.

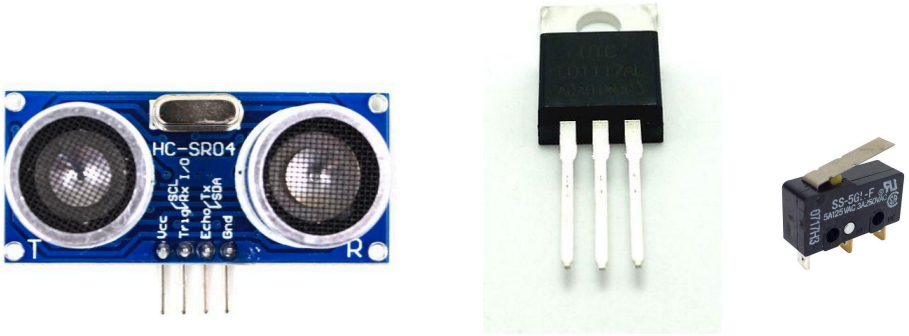

Figure 21. Ultrasonic sensor, voltage regulator and limit switch

4. Connection with Scissor Lift ESP32
   The AGV had an infrared transmitter that also turned on when the limit switch was pushed. For this we used an infrared sensor and we covered the receiver with insulation tape.

Figure 22. Infrared sensor used as transmitter only

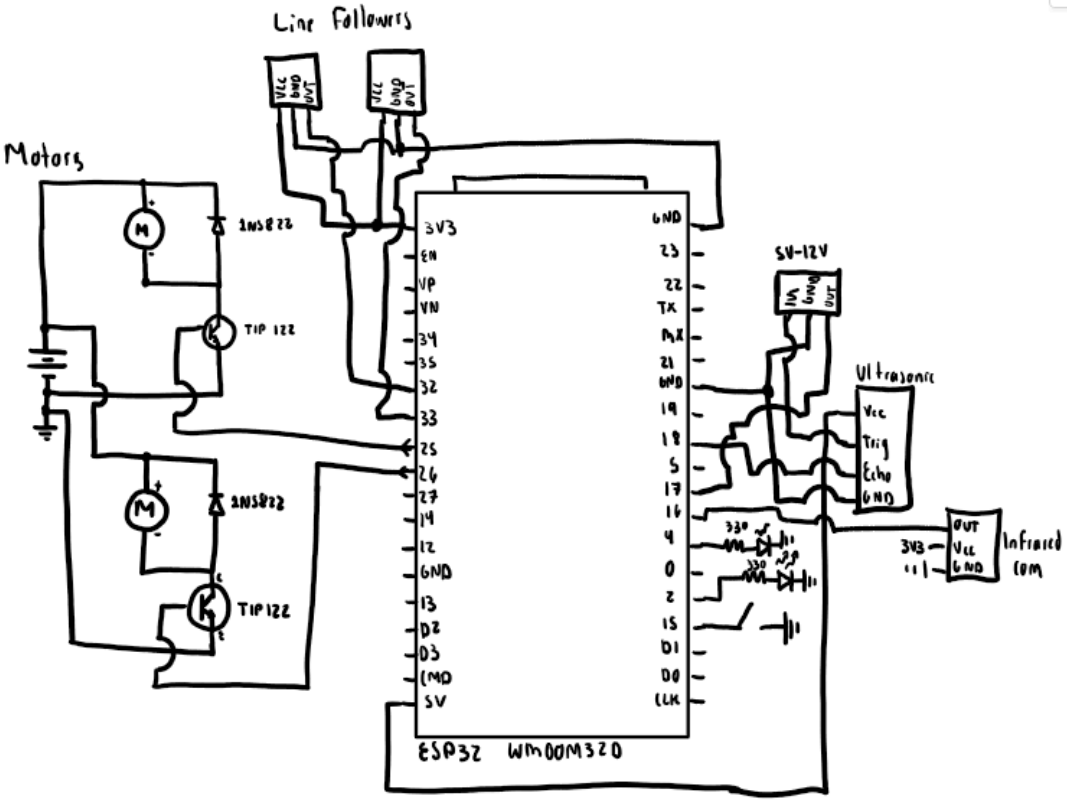

Figure 23. Schematic Circuit of AGV

Scissor Lift

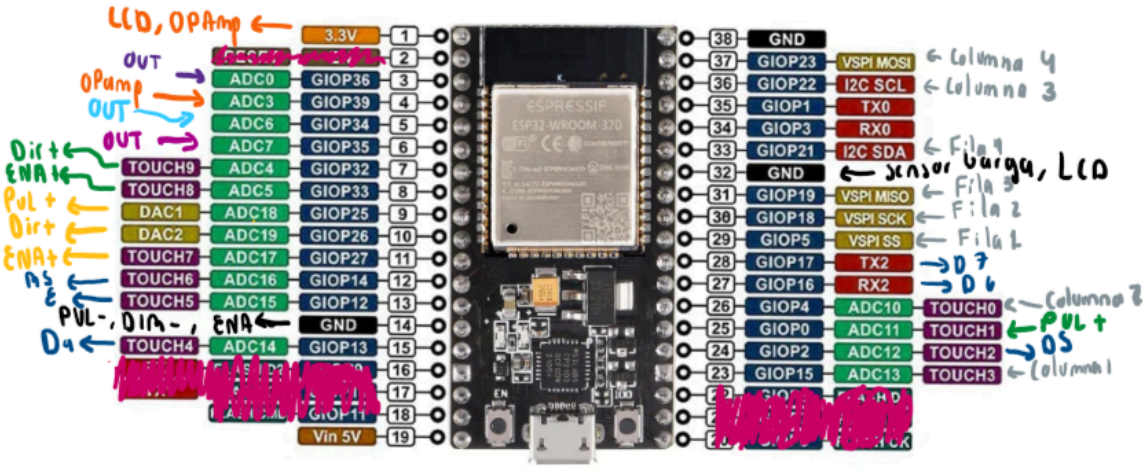

Figure 24. ESP32 connections for Scissor Lift

The systems that conform the scissor Lift are:

1. Lifting mechanism

   For the lifting mechanism we used a Stepper motor Nema 23 of 12 V, 3.5 A and 20 Kgcm, this motor was controlled by an TB6600 driver, we used this driver because it allowed us to modify the current given to the motor and it was easier to set up.

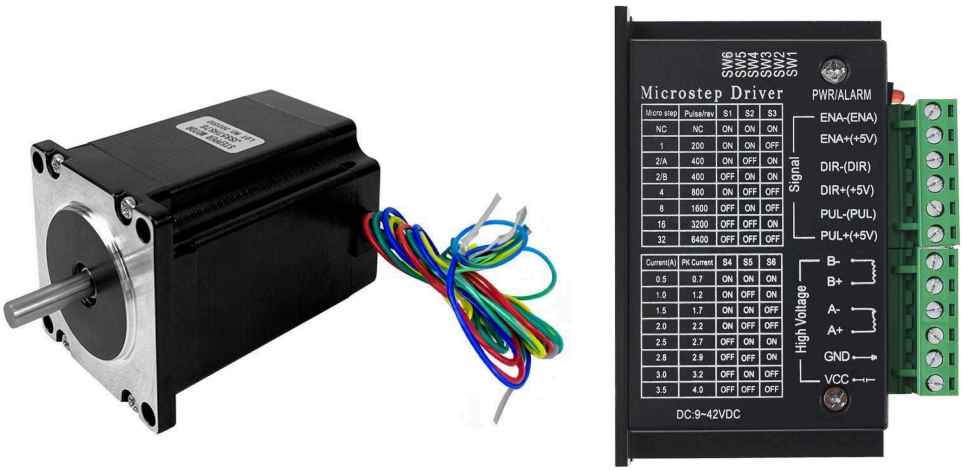

Figure 25: Stepper Motor and driver

2. Tilting mechanism

   For the lifting mechanism we used a Stepper Motor Nema 23 of 12V, 2.8 A and 12.8 Kgcm, this motor was controlled by an TB6600 driver, we used this driver because it allowed us to modify the current given to the motor and it was easier to set up.

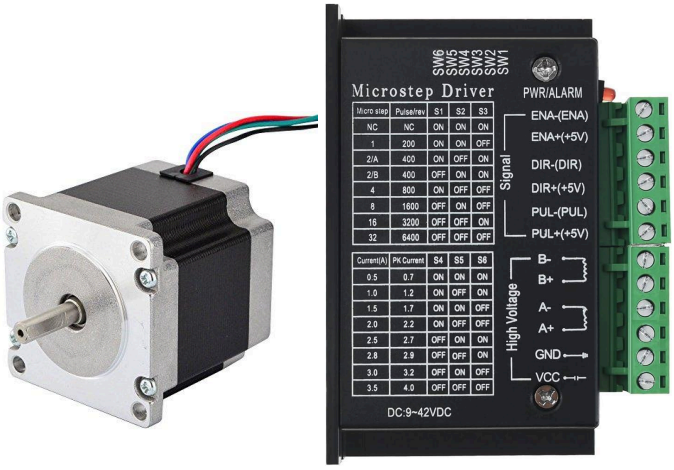

Figure 26: Stepper Motor and driver

3. LCD
   A generic LCD was used to show the progress made by the system.

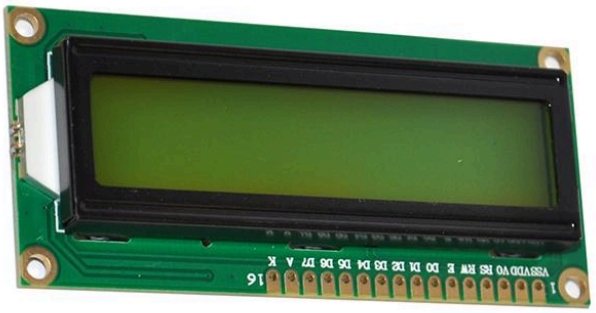

Figure 27: LCD screen

4. Keypad
   A matrix keypad was used to set the weight limit for the load.

Figure 28: Keypad

5. Load sensor
   A load cell was used to weight the load of the beans, as we couldn't use a HX711 module we had to implement a circuit of Operational Amplifiers to make it work properly.

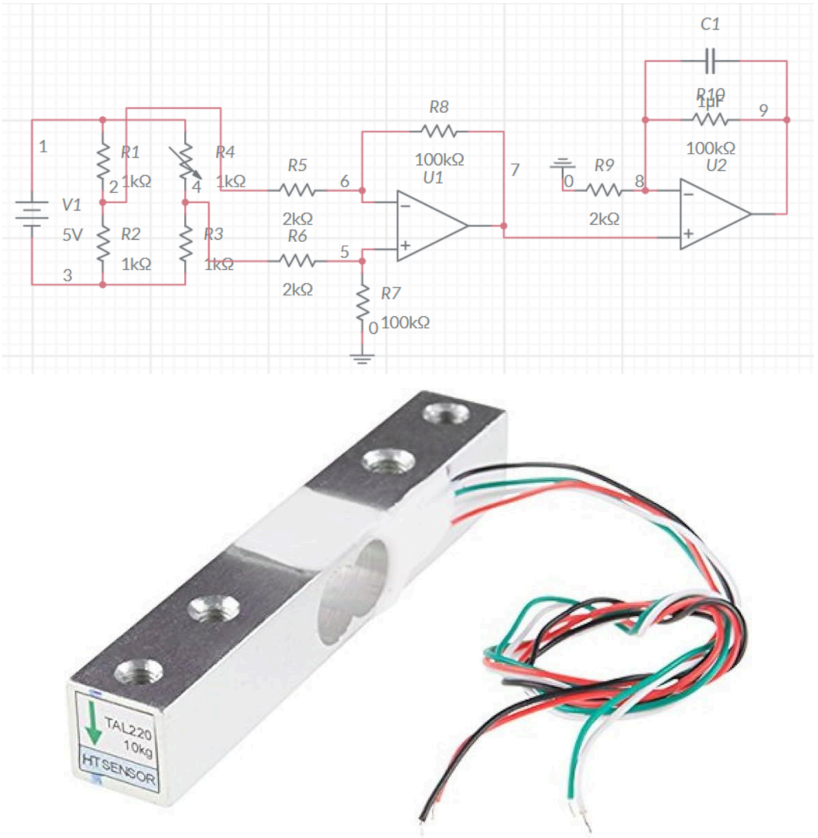

Figure 29. Load Cell and Circuit

6. Height sensor
   Knowing that there was going to be a strip of infrared Leds to mark the wanted height we used an infrared sensor to detect the strip, we only had to tape the transmitter so that there wouldn't be any problem with unwanted signals.

Figure 30. Infrared sensor as receiver

7. Connection with AGVs ESP32
   The Scissor Lift had an infrared receiver that received the infrared light from the AGV, with this, the Scissor Lift was able to know when to start working. For this we used an infrared sensor and we covered the transmitter with insulation tape.

Figure 31. Infrared sensor as receiver

8. Servo Motor for the Gate.
   The gate was open with a Servo Motor controlled by the ESP 32

Figure 32. Servomotor

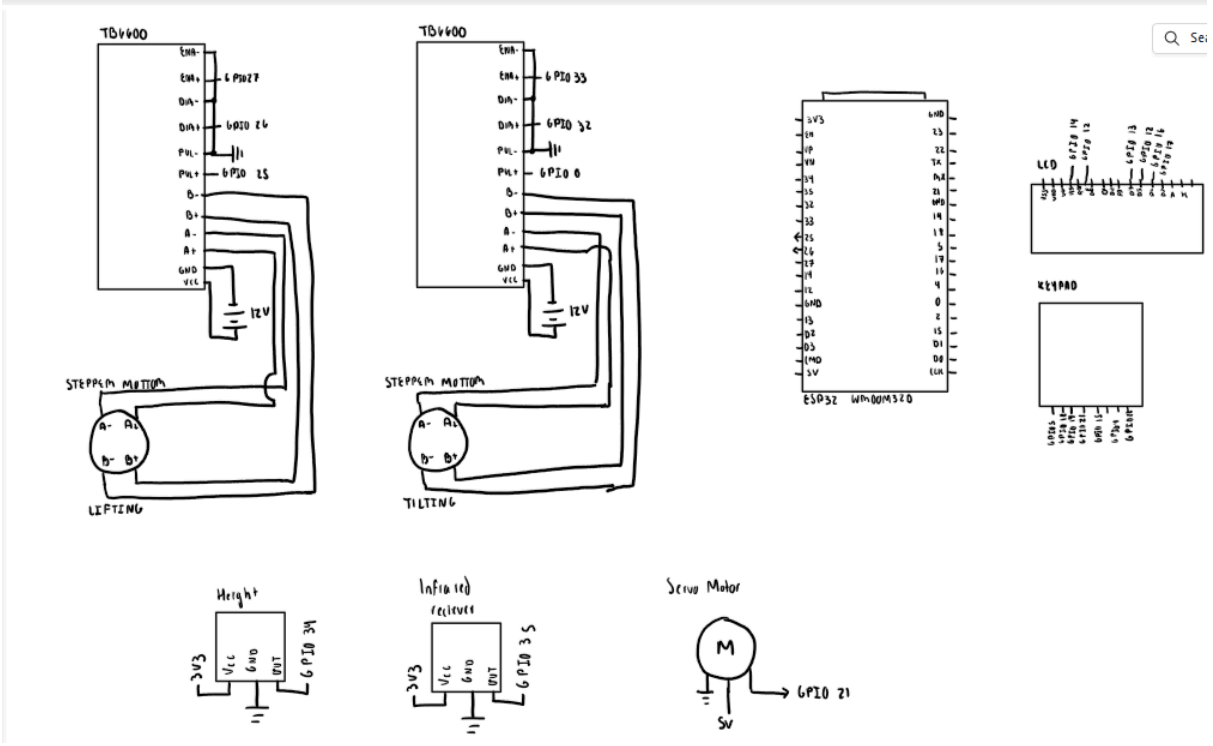

Figure 33. Schematic Circuit of Scissors Lift

Videos of the system:
https://www.canva.com/design/DAGqeZrAG2U/K2I8yF3Fd7qhfoSr7b59gA/view?utm_content=DAGqeZrAG2U&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=h34e9c077a6

INSTRUMENTATION

Theoretical and practical considerations involved in interfacing a load cell sensor with an ESP-32 microcontroller.

The load cell utilized is a bar-type sensor, primarily constructed from an aluminum-alloy. This material is selected for its mechanical properties, which allow it to undergo precise and repeatable deformation under applied mechanical force.

Internally, the load cell incorporates four strain gauges (variable resistors), attached to its surface. These strain gauges are designed to exhibit a change in their resistance directly proportional to the deformation they experience. The configuration of these strain gauges forms a Wheatstone bridge circuit.

The Wheatstone bridge helps us to measure the difference in voltage between two terminals. When all four resistors within the bridge are equal, the voltage difference across its midpoints is ideally zero, indicating a balanced state. Upon the application of force to the load cell, the physical deformation induces a change in the resistance of the strain gauges. This change unbalances the Wheatstone bridge, generating a measurable differential voltage output across its signal terminals. This output voltage is directly correlated with the magnitude of the applied force.

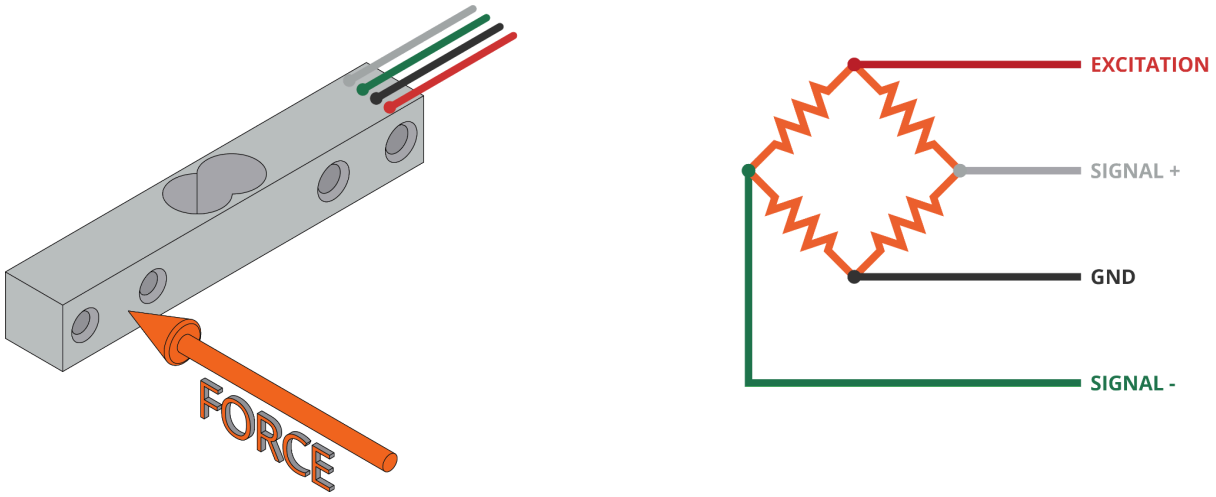

Figure 34. Load cell's wheatstone bridge functionality. (MonoDAQ. 2020)

Unfortunately, voltage output from the Wheatstone bridge is typically very small, often in the millivolt range. To effectively interface this signal with the ESP-32, which operates with higher voltage input ranges, amplification was needed.

Commercial load cell kits frequently include dedicated signal conditioning modules, such as the HX711. The primary function of this module is to amplify the minute differential voltage from the load cell and convert it into a digital signal suitable for direct microcontroller interfacing.

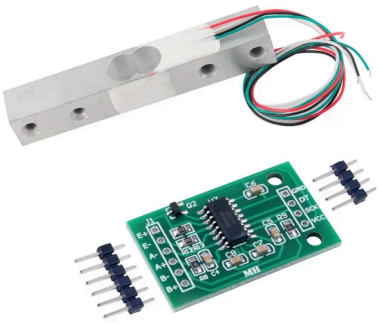

Figure 35. Kit that is widely used

Without access to pre-built drivers like the HX711, a challenge was designing a custom op-amp based amplification circuit. This circuit needed to amplify the load cell's small output voltage to match the ESP-32's optimal input range when measuring up to 5kg, requiring component selection and gain calculation.

Based on a circuit prepared by the professor during the laboratory practice, we gained a clear understanding of how the circuit for amplifying the output voltage should be configured.

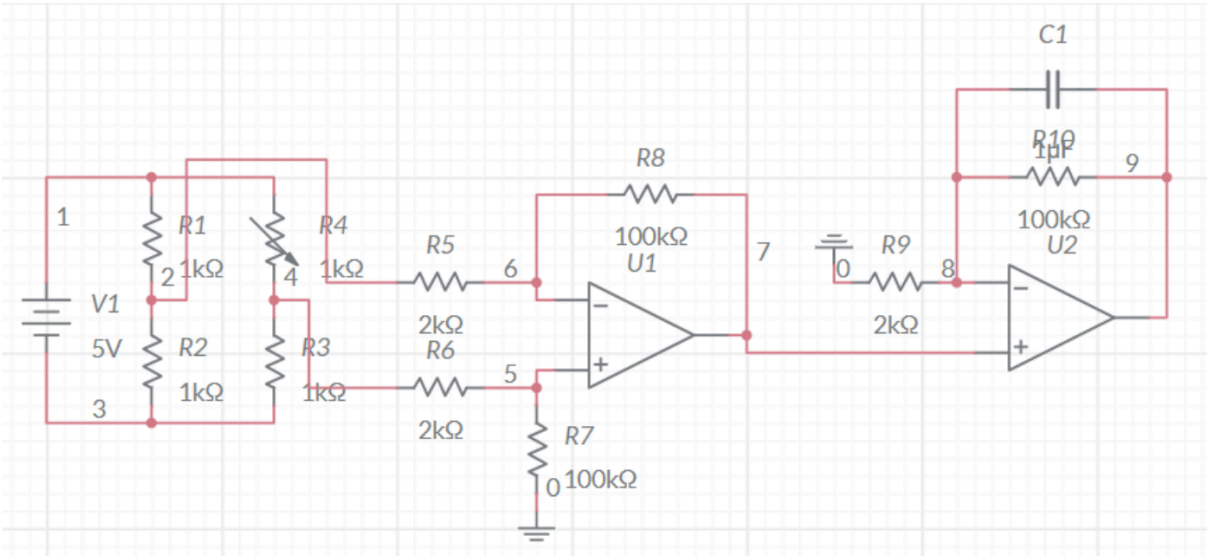

Figure 36. Final circuit design.

By following the next formulas and relations we could calculate the amount of times the output voltage would be bigger.

In differential amplifier phase:

$$R_5 = R_6\,,\ R_8 = R_7$$

$$V_0 = \frac{R8}{R5}(V_1 - V_2) \rightarrow V_0 = \frac{100}{2}(V_1 - V_2) \rightarrow V_0 = 50\,\Delta V$$

The amplification factor of the output voltage is directly proportional to the ratio of the higher resistance to the smaller resistance; in this case, after passing through the differential amplifier, the output would be fifty times its original value.

In non-inverting amplifier phase:

$$R_{in} = 2k\Omega \, , R_f = 100k\Omega$$

$$V_0 = (1 + \frac{Rf}{Rin})V_{in} = (1 + \frac{100}{2})V_{in}$$

$$V_0 = 51V_{in}$$

Total amplification: $V_0 = 2550\,\Delta V$

After the calculations we were ready to build the circuit and connect it to a voltage source.

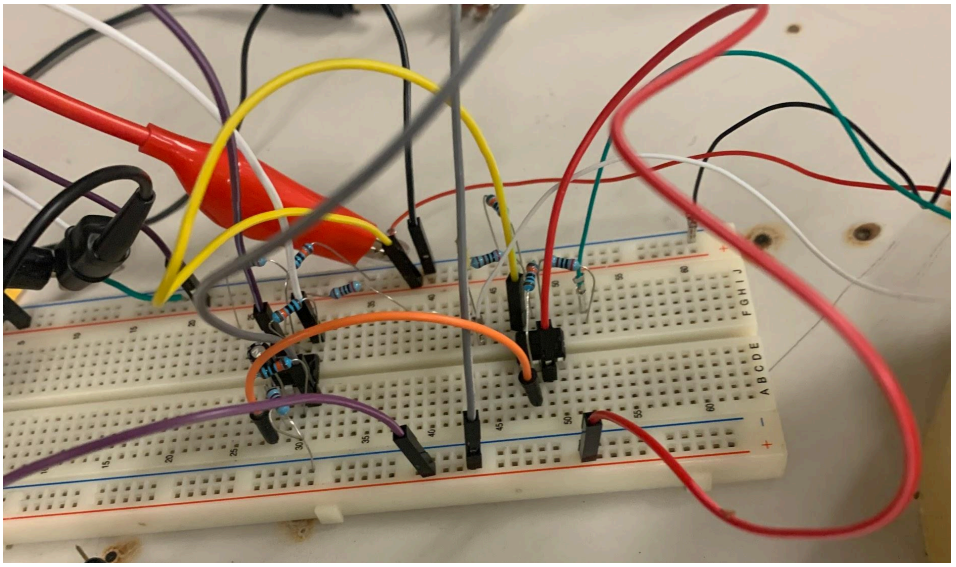

Figure 37. Physical implementation of the circuit

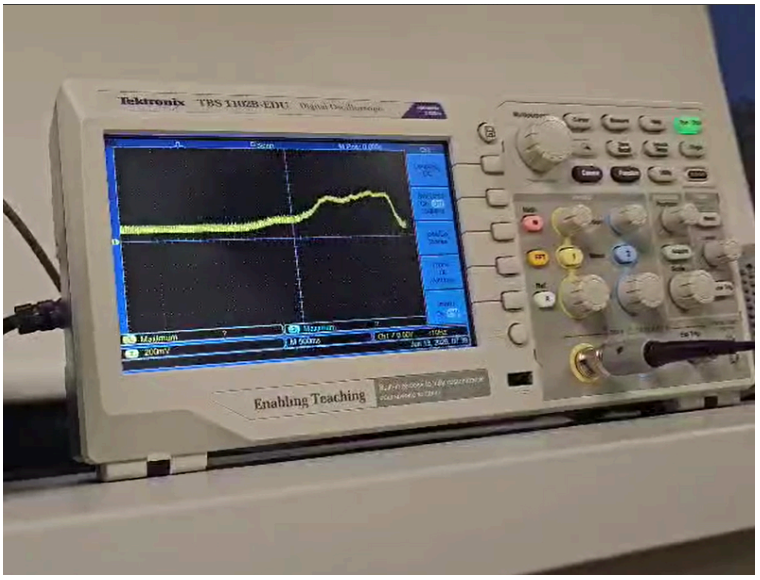

Figure 38. Oscilloscope output exhibiting changes due to strain gauge deformation

The circuit was successfully assembled and tested using 5 kg gym weights to apply force, which in turn altered the strain gauges' resistance. Despite utilizing two amplification stages, one configured for a gain of 50 and the other of 51, the peak output voltage observed was

2V during manual pressure and 460 mV under the specified weight load (5Kg). Adhering to design guidelines, we avoided further increasing the amplification factor, even after considering an initial signal offset of -0.4 V.

INDIVIDUAL CONCLUSIONS

Leonardo Valero Perales

The task of designing, manufacturing, and implementing a scissor lift is a grueling one. Plenty of things can and will go wrong, nevertheless we must push forward if we are to develop our skills as mechatronic engineers. The greatest challenge was the design and implementation of the electronic circuits and sensors that allowed the systems to see the world around them, and the microcontrollers that read and transformed their signals into useful data that we could then act upon.

Another challenge was that of ensuring the smooth operation of mechanical systems mainly found in the lift, but also in certain parts of the AGV like the wheels which at first would freely spin without moving the vehicle.

However, I must admit that I felt disappointed in what felt like a lack of guidance from our professors in the sense that what subjects we touched upon in class had little to do with the project, or when they did, we were not taught how to implement them in the project. Don't get me wrong, what we learnt was definitely useful for the future, but it wasn't clear how learning about diodes for 5 weeks would help us in designing circuits from scratch. We weren't taught how to write new code, but instead how to rewrite an already existing one. I don't blame our professors; instead I believe it has to do with the fundamental incompatibility of the Tec 21 model and the engineering philosophy. During certain strecthes of the project I remembered a quote I learnt that students from MIT tell to one another: "Learning at MIT is like drinking water from a firehose. Most may drown, but those who don't will never know thirst again". I believe this project could have been one where we all drank from that same hose, but instead the hose we drank from was turned halfway off and it was also the wrong one.

Ultimately, even if we failed in creating an AGV and scissor lift that worked when all their parts were fully implemented, I'd still consider this project a success as the amount of new knowledge we had to obtain was massive and while we suffered a few sleepless nights I still consider the experience a delightful one for my conviction is unwavering.

Frida Sophia Chávez Juárez:

This project was both intellectually stimulating and technically demanding. It challenged me to explore and learn independently about essential components such as sensors, actuators, and microcontrollers. Through hands-on experimentation and problem-solving, I developed a much stronger understanding of how these technologies function and interact within a system.

However, despite these valuable learning experiences, I believe that we needed more time to fully comprehend certain complex topics particularly state machines and the integration of all system elements. The project required us to understand and apply a wide range of concepts in a relatively short amount of time, which made it difficult to go beyond surface-level knowledge and truly master the material. In particular, the implementation

phase was the most challenging, as we lacked sufficient instruction or structured guidance in how to bring all components together into a fully functioning system.

Although the final result did not meet our expectations, the experience was far from a failure. We learned a great deal through our mistakes, and these lessons will be incredibly valuable moving forward. Recognizing what went wrong, analyzing the causes, and reflecting on how to improve are all critical steps in the learning process. I'm confident that the knowledge gained from this project, especially the setbacks, will help me avoid similar issues in future projects and make me a more effective engineer in the long term.

Maximiliano Elizarraras Shchelkunova:

Personally speaking, I was somewhat disappointed by the project as a whole. Even if it succeeded in the end, I don't feel that I have improved in all the areas that I should have as an engineer, let alone a mechatronics one who is supposed to know about different aspects of a real project. Due to the project's complexity paired with a questionable education model, the objective of learning was not completed. Every member in each of the teams became responsible for a particular area, instead of learning about all of them at once. I may have become more proficient at mechanics and Oscar at programming, but not vice versa. That is the reason why I prefer either taking the whole responsibility of a project or doing individual and smaller projects. I know that it most probably won't be possible, but just in case, I would like to plead you to alter future projects, focusing mainly on individual performance. I know working in a team is important, but in order to work in a team in the real world, one must have acquired all the necessary skills beforehand, which is not being done. As for the specific objectives of the project, Most of them were accomplished for the final product. The motors moved and the sensors detected. Even if in the end they did not function together, individually they accomplished their function.

Victor Manuel Galicia Hernández

This project took a big stride in comparison to other semesters' final deliverables. We actually built something that genuinely qualifies as mechatronic, and while that's a huge win, it also brought us face to face with failure. It was a great dose of reality, reminding us that we're still just beginners.
But the most incredible aspect of experiencing failure is the vast potential for growth it reveals. I hope that with better preparation, time management and dedication, the next project will be nothing short of a resounding success.

Oscar Gadiel Ramo Martínez

Although the project didn't conclude as we hoped, this semester was full of valuable learning. We deepened our understanding not only of AGVs and scissor lifts, but also of the fundamental concepts that form the backbone of our career. Working across all the areas of mechatronics was challenging, but it marked the beginning of our journey as future engineers.
Each subsystem succeeded in isolation, yet we found that integration was the most demanding—and ultimately, where we struggled. Still, I carry forward the experience and

lessons gained. The insights we've drawn from our missteps will be essential in future projects and in our development as professionals. This project wasn't just a task—it was a stepping stone.

REFERENCES

- Mark Harris (2020). An Introduction to Wheatstone Bridge Circuits and Differential Amplifiers. https://resources.altium.com/p/wheatstone-bridges
- Soldered Electronics (n.d.) . Hx711 - How it works. https://soldered.com/documentation/hx711/how-it-works/#:~:text=How%20the%20HX711%20works%E2%80%8B,measurements%20with%20minimal%20noise%20interference.
- Allen, P.;& Holdberg R. D., CMOS analog circuit design, 3a, USA : OUP USA, 2012
- Bariáin, A. C.;& Corres, S. J. , Programación de Microcontroladores PIC en lenguaje C, 1a, Alfaomega, 2017, Español
- Budynas, R. G., Nisbett, J. K.; & Shigley, J. E. , Shigley's mechanical engineering design , 10a, McGraw-Hill Education, 2015
- Bolton W, Mecatrónica : sistemas de control electrónico en la ingeniería mecánica y eléctrica, 5a ed., México : Alfaomega, 2013
- Boylestad, R. L., & Nashelsky, L. (2007). *Electronic Devices and circuit Theory*.

- freeCodeCamp.org. (2023, October 24). Harvard CS50 (2023) – Full Computer Science University course [Video]. YouTube. https://www.youtube.com/watch?v=LfaMVlDaQ24
- LaBuhardillaDelLoco. (2022, January 25). 🔴 ESP32 Salidas PWM [Video]. YouTube. https://www.youtube.com/watch?v=azFFLbQvsRc
- Jouaneh Musa, Fundamentals of mechatronics, Stamford, CT : Cengage Learning, 2013

ANNEX SECTION

Annex (Matlab Pinion Rack Code)

```
%Initial Variables
Dist = 0.25; %Radius of the Pinion
Weight = 6.5; %Weight
Speed = 0.05; %Possible Speed
Diam = 0.05; %Diameter of the Pinion

%Calculating Forces
Fy = (Weight*0.14*9.81)/Dist;
FyT = Fy + (Weight/2)*Speed

% Final torque (in kg/cm) necessary
Torque = (FyT*Diam) * 10.19716
```

Annex (AGV main Code)

```
#include <definitions.h>

// Constant definitions
#define MIN_DISTANCE 30 // cm
#define MAX_DISTANCE 10 // cm
#define SOUND_AIR_SPEED 343 // m/s
```

```
enum states {state0, state1, state2};

// SUPPORT-FUNCTIONS
// Line Follower
bool lineFollowerLogic(int a, int b) {
    switch ((a << 1) | b) {
        case 0b00: // Both sensors off
            dcMotor_1.setDuty(00);
            dcMotor_2.setDuty(00);
            comSensorObstacleLogic(3);
            return true;
            break;
        case 0b01: // Left On, Right Off
            dcMotor_1.setDuty(25);
            dcMotor_2.setDuty(75);
            return false;
            break;
        case 0b10: // Left Off, Right On
            dcMotor_1.setDuty(75);
            dcMotor_2.setDuty(25);
            return false;
            break;
        case 0b11: // Both sensors on
            dcMotor_1.setDuty(50);
            dcMotor_2.setDuty(50);
            return false;
            break;
    }

}

// Collision Avoidance
float read_distance(SimpleGPIO &trig, SimpleGPIO &echo) {
    int64_t start_time = 0, end_time = 0, delta_time;
    float distance;
    // Activate trig
    trig.set(0);
    ets_delay_us(2); // Short delay
    trig.set(1);
    ets_delay_us(10); // 10 us high pulse
    trig.set(0);
    // Echo readings
```

```
        while (echo.get() == 0);
        start_time = esp_timer_get_time(); //Wait for echo to go high
        while (echo.get() == 1);
         end_time = esp_timer_get_time(); // Wait for echo to go low
(signal complete)
        delta_time = end_time - start_time;
           if (delta_time > 0) distance = (delta_time * 1e-6 *
SOUND_AIR_SPEED * 100) / 2; // us to seconds
        else return -1; // Invalid distance
        return distance;

    }

    void collisionAvoidanceLogic(float distance) {
        const float m = 50/(MIN_DISTANCE - MAX_DISTANCE);
        const float b = -m * MAX_DISTANCE;
        int percentage = static_cast<int>(m * distance + b);
           percentage = std::clamp(percentage, 0, 100); // Ensure
percentage is within 0-100
        dcMotor_1.setDuty(percentage);
        dcMotor_2.setDuty(percentage);
    }

    // Communication Sensor
    void comSensorObstacleLogic(int com_State, bool obstacleDetected
= false) {
        // Communication sensor blink variables
        static bool blinkState = false; // Blink state
        static int64_t lastBlink = 0; // Last blink time
        int64_t now;
        switch (com_State) {
            case 1:
                agvComSensor.set(1);
                break;
            case 2:
                   now = esp_timer_get_time() / 1000; // Get current
time in milliseconds
                if (obstacleDetected && (now - lastBlink > 200)) {
                    blinkState = !blinkState; // Toggle blink state
                    agvComSensor.set(blinkState); // Toggle signal
                    lastBlink = now; // Update last blink time
                }
```

```
                                else if (obstacleDetected == false)
agvComSensor.set(1);
                    break;
                case 3:
                    agvComSensor.set(0);
                    break;
            }
        }

        // Function for led blinking
        void ledBlink(SimpleGPIO &sensor, int repetition, int duration) {
            for (int i = 0; i < repetition; i++) {
                                                        sensor.set(1);
// Turn on sensor
                                    vTaskDelay(pdMS_TO_TICKS(duration));
// Wait for 1 second
                                                        sensor.set(0);
// Turn off sensor
                                    vTaskDelay(pdMS_TO_TICKS(duration));
// Wait for m1 second
            }
        }

        // MAIN FUNCTIONS
        bool setup() {
                lineFollower_1.setup(LINE_FOLLOWER1_GPIO, GPI); // GPIO,
input mode, default pull
                lineFollower_2.setup(LINE_FOLLOWER2_GPIO, GPI); // GPIO,
input mode, default pull
              dcMotor_1.setup(DCMOTOR1_GPIO, 0); // GPIO, channel, else =
default setup
              dcMotor_2.setup(DCMOTOR2_GPIO, 1); // GPIO, channel, else =
default setup
             colliAvoidance_1_trig.setup(COLL_AVOIDANCE1_TRIG_GPIO, GPIO);
// GPIO, output mode, default pull
              colliAvoidance_1_echo.setup(COLL_AVOIDANCE1_ECHO_GPIO, GPI);
// GPIO, input mode, default pull
               agvComSensor.setup(COMM_SENSOR_GPIO, GPIO); // GPIO pin,
input mode, default pull
               greenLed.setup(GREEN_LED_GPIO, GPO); // GPIO pin, output
mode, default pull
              redLed.setup(RED_LED_GPIO, GPO); // GPIO pin, output mode,
default pull
```

```
            golpeAvisa.setup(GOLPE_AVISA_GPIO, GPI); // GPIO pin, input
mode, default pull
          return true;
      }

      int move_agv(int agv_state) {
          // Variables defined
          float distance;
          bool obstacleDetected = false;
          bool read_collision;
          bool exit;
          // AGV moving state
          switch (agv_state) {
              case 1:
                  read_collision = false;
                  comSensorObstacleLogic(1);
                  break;
              case 2:
                  read_collision = true;
                  break;
          }
          // Initialize motors
          dcMotor_1.setDuty(50); // Duty percentage
          dcMotor_2.setDuty(50); // Duty percentage
          // Infinite loop
          while(true) {
              // Readings
                          int   a   =   lineFollower_1.get();   //int   a   =
gpio_get_level((gpio_num_t)LINE_FOLLOWER1_GPIO);
                          int   b   =   lineFollower_2.get();   //int   b   =
gpio_get_level((gpio_num_t)LINE_FOLLOWER2_GPIO);
              int c = golpeAvisa.get();
                           if   (read_collision   ==   true)   distance   =
read_distance(colliAvoidance_1_trig, colliAvoidance_1_echo);
              // Infrarred sensors
              exit = lineFollowerLogic(a, b);
              if (exit == true) return 1;
              // Collision Avoidance Sensors
               if (distance <= MIN_DISTANCE && distance >= MAX_DISTANCE)
{
                  printf("Obstacle detected! At %.2f\n", distance);
                  collisionAvoidanceLogic(distance);
                  obstacleDetected = true;
```

```
                comSensorObstacleLogic(2, obstacleDetected);
            }
            else if (distance > MIN_DISTANCE) {
                    printf("No obstacle nearby! Distance is %.2f\n",
distance);
                obstacleDetected = false;
                comSensorObstacleLogic(2, obstacleDetected);
            }
            else obstacleDetected = false;
            // Switch button
            if (c == 1) return c;
            vTaskDelay(pdMS_TO_TICKS(500));
        }
    }

    extern "C" void app_main() {
        int next_state;
        bool good;
        states state = state0;
        for (int i = 0; i < 3; i++) {
            switch (state) {
                case state0: // Setup all components
                    good = setup();
                    if (good == true) {
                        ledBlink(greenLed, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        break;
                    }
                    else {
                        ledBlink(redLed, 1, 2000);
                        exit(0);
                    }
                case state1: //Move AGV without collision sensors
                    next_state = move_agv(1);
                    if (next_state == 1) {
                        ledBlink(greenLed, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        break;
                    }
                    else exit(0);
                case state2: //Move AGV without collision sensors
```

```
                    next_state = move_agv(2);
                    if (next_state == 1) {
                        ledBlink(greenLed, 1, 1000);
                        exit(0);
                        break;
                    }
                }
            }
        }
```

Annex (AGV definitions.h Code

```
    #ifndef _DEFINITIONS_H_
    #define _DEFINITIONS_H_

    //Libraries to use
    #include <cstdlib>                  // To end program
    #include <rom/ets_sys.h>             // Delay without interrumping
all the program
    #include <SimpleADC.h>              // Analog signals
    #include <SimpleGPIO.h>             // Digital signals
    #include <SimplePWM.h>              // Motors
    #include <SimpleTimer.h>            // Control time
    #include <algorithm>                // Process data

    //GPIO pins
    //  DC motor
    #define DCMOTOR1_GPIO 25
    #define DCMOTOR2_GPIO 26
    //  Line follower
    #define LINE_FOLLOWER1_GPIO 33
    #define LINE_FOLLOWER2_GPIO 32
    //  Collision avoidance
    #define COLL_AVOIDANCE1_TRIG_GPIO 17
    #define COLL_AVOIDANCE1_ECHO_GPIO 18
    //  Comm sensor
    #define COMM_SENSOR_GPIO 16
    // LEDs to indicate phases
    #define GREEN_LED_GPIO 2 // Check with teammate the number
    #define RED_LED_GPIO 4 // Check with teammate the number
    // Switch button to change agv state
```

```
#define GOLPE_AVISA_GPIO 15

//Object creation
//  DC Motor
SimplePWM dcMotor_1;
SimplePWM dcMotor_2;
//  Line follower
SimpleGPIO lineFollower_1;
SimpleGPIO lineFollower_2;
//  Collision avoidance
SimpleGPIO colliAvoidance_1_trig;
SimpleGPIO colliAvoidance_1_echo;
//  Communication sensor
SimpleGPIO agvComSensor;
// LEDs
SimpleGPIO greenLed;
SimpleGPIO redLed;
// Button
SimpleGPIO golpeAvisa;

#endif // _DEFINITIONS_H_
```

Annex (Scissor lift main Code)

```
#include <definitions.h>

enum states {state0, state1, state2, state3, state4, state5,
state6};

// SUPPORT FUNCTIONS
//LED Actuator
void blinkLED(SimpleGPIO &sensor, int repetition, int ms = 200) {
    for (int i = 0; i < repetition; i++) {
        sensor.set(1);                                  // Turn
on sensor
         vTaskDelay(pdMS_TO_TICKS(ms));                 // Wait
for ms
```

```
            sensor.set(0);                                          // Turn
off sensor
             vTaskDelay(pdMS_TO_TICKS(ms));                         // Wait
for ms
         }
     }

     // Lift callback function
     void IRAM_ATTR liftCallback(void* arg) {
         static bool state = false;
         state = !state;
          liftPul.set(state);                                       // Toggle
the pulse signal for the lift motor
     }

     // Tilt callback function
     void IRAM_ATTR tiltCallback(void *arg) {
         static bool state = false;
         state = !state;
          tiltPul.set(state);                                       // Toggle
the pulse signal for the tilt motor
     }

     // Tilt stop callback function
     void IRAM_ATTR tiltStopCallback(void* arg) {
          tiltTimer.stopPeriodic();                                 // Stop
generating steps
          tiltEna.set(1);                                           //
Disable motor (if 1 = disable on your driver)
     }

     // Keypad
     float keypadLogic() {
         // Setup
         lcdDisplay.setup(lcd_pins);
         keypad.setup();
          char buffer[3] = {'\0'};                                  // Buffer
to store the input weight
          int index = 0;                                            // Index
for the buffer
         // Input weight from user
         lcdDisplay.printStr("Input load\nweight in kg");
         vTaskDelay(pdMS_TO_TICKS(3000));
```

```
          lcdDisplay.printStr("Press 'A' to\nconfirm");
          while(true) {
              char key = keypad.getKey();
              if (key != '\0') {
                          if  (key  >=  '0'  &&  key  <=  '9'  &&  index  <
sizeof(buffer) - 1) {
                        buffer[index++] = key;                    // Store
the digit in the buffer
                          buffer[index] = '\0';                        //
Null-terminate the string
                      lcdDisplay.writeCommand(CMD_CLEAR);
                      lcdDisplay.printStr(buffer);
                  }
                  else if (key == 'A' && index > 0) {
                        return atof(buffer);               // Convert to
float and return
                  }
                  else if (key == 'C') {
                      lcdDisplay.writeCommand(CMD_CLEAR);
                        buffer[0] = '\0';                            // Clear
the buffer
                        index = 0;                                   // Reset
index
                  }
                  else if (key == 'B' && index > 0) {
                          index--;                                        //
Decrement index
                       buffer[index] = '\0';                     // Remove
last character
                      lcdDisplay.writeCommand(CMD_CLEAR);
                      lcdDisplay.printStr(buffer);
                  }
              }
              vTaskDelay(pdMS_TO_TICKS(200));
          }
      }

      // Load Cell
      void loadCellLogic() {
          // Variables defined
           float inputWeight;                                       // User
input weight
```

```
          float reads;                                                    //
Variable to store the load cell reading
          const float m = 0.1, b = 0.1;                                   //
Calibration constants
          float realWeight;                                        // Real
weight from load cell
          float lastPrintedWeight = -1000;                         // Last
printed weight to avoid flickering
          int stableCount = 0;                                            //
Counter for stable weight readings
          char msg[32];                                           // Buffer
for messages
          inputWeight = keypadLogic();
          lcdDisplay.printStr("Loading beans\nPlease wait...");
          while(true) {
               reads = loadCell.read(ADC_READ_MV);                 // Read
load cell value
               realWeight = m*reads + b;                           // Real
weight calculation
              //Show weight only if it changed
              if (fabs(realWeight - lastPrintedWeight) > 0.05f) {
                  sprintf(msg, "Current Weight:\n%.2f kg", realWeight);
                  lcdDisplay.printStr(msg);
                   lastPrintedWeight = realWeight;                 // Update
last printed weight
              }

              // Check if weight is stable
              if (fabs(realWeight - inputWeight) < 0.05f) {
                  stableCount++;
              }
              else {
                   stableCount = 0;                                // Reset
stable count if weight is not stable
              }
                if (stableCount >= 2) {                                 // 2
second stability check
                   ledAct.set(1);                                  // Turn
on buzzer
                  lcdDisplay.printStr("Load weight\nreached!");
                   vTaskDelay(pdMS_TO_TICKS(3000));                // Wait
for 3 seconds
```

```
                    ledAct.set(0);                                  // Turn
off buzzer
                    break;                                          // Exit
the loop)
                }
                 vTaskDelay(pdMS_TO_TICKS(1000));                   // Wait
for 1 sec before next reading
            }
        }

        // Scissor Lift Communication Sensor
        bool comSensorDetect(SimpleGPIO &sensor, int expectedState, int
duration_ms, const char *msg, int interval_ms = 50) {
        int read; // Variable to store comm sensor reading
        int64_t now; // Variable to store current time
        int64_t startTime = 0; // Variable to store start time of
detection
        while(true) {
            read = sensor.get(); // Comm sensor reading
            now = esp_timer_get_time() / 1000; // Current time in ms
            if (read == expectedState) {
                if (startTime == 0) { // First detection
                    startTime = now; // Start detection time
                }
                   else if (now - startTime >= duration_ms) { // Signal
stable for specified duration
                    lcdDisplay.printStr(msg);
                    return true; // Successful detection
                }
            }
            else startTime = 0; // Reset timer if signal is lost
                 vTaskDelay(pdMS_TO_TICKS(interval_ms)); // Check every
interval
        }
        return false; // Detection failed
        }

        // MAIN FUNCTIONS
        bool setup() {
            // LCD
             lcdDisplay.setup(lcd_pins);                            // LCD
pins
            lcdDisplay.printStr("System Initializing...");
```

```
          // Servomotor
           servoMotor.setup(SERVOMOTOR_GPIO, 0);                  // GPIO
pin, channel, rest = default
          // Tilt Stepper motor
           tiltPul.setup(TILT_PUL_GPIO, GPO);                     // GPIO
pin, output mode, default pull
           tiltDir.setup(TILT_DIR_GPIO, GPO);                     // GPIO
pin, output mode, default pull
           tiltEna.setup(TILT_ENA_GPIO, GPO);                     // GPIO
pin, output mode, default pull
            tiltDir.set(1);                                         //
Direction for tilt motor
           tiltEna.set(1);                                        // tilt
motor off
          tiltTimer.setup(tiltCallback, "tilt_timer");
          tiltStopTimer.setup(tiltStopCallback, "tilt_stop_timer");
          // Lift Stepper Motor
           liftPul.setup(LIFT_PUL_GPIO, GPO);                     // GPIO
pin, output mode, default pull
           liftDir.setup(LIFT_DIR_GPIO, GPO);                     // GPIO
pin, output mode, default pull
           liftEna.setup(LIFT_ENA_GPIO, GPO);                     // GPIO
pin, output mode, default pull
            liftDir.set(1);                                         //
Direction for lift motor
           liftEna.set(1);                                        // Lift
motor off
          liftTimer.setup(liftCallback, "lift_timer");
          // Other components
           heightSensor.setup(HEIGHT_SEN_GPIO, GPI);              // GPIO
pin, input mode, default pull
           loadCell.setup(LOAD_CELL_GPIO);                        // GPIO
pin, default width = bit 12
           ledAct.setup(BUZZER_GPIO, GPO);                        // GPIO
pin, output mode, default pull
          keypad.setup();
           slComSensor.setup(COMM_SENSOR_GPIO, GPI);              // GPIO
pin, input mode, default pull
          return true;
      }

      bool load_beans() {
          keypadLogic();
```

```
          // loadCellLogic();
          return true;
      }

      bool waiting_agv() {
          char msg[] = "AGV coupled succesfully!\nMoving mechanism...";
          lcdDisplay.printStr("Waiting for AGV\nto couple...");
          while(true) {
                 return comSensorDetect(slComSensor, 1, 3000, msg); //
Detect if mechanism is fully coupled
          }
          return true;
      }

      bool move_mechanism() {
          char msg_1[] = "Obstacle detected!";
           char msg_2[] = "The mechanism has arrived at the unloading
station!";
          lcdDisplay.printStr("Moving to unload\nstation...");
          while(true) {
               comSensorDetect(slComSensor, 0, 200, msg_1); // Detect if
mechanism encounters an obstacle
                return comSensorDetect(slComSensor, 0, 3000, msg_2); //
Detect if mechanism arrives at unloading station
          }
          return true;
      }

      bool lifting_motor() {
          // LCD Setup
          lcdDisplay.setup(lcd_pins);
           heightSensor.setup(HEIGHT_SEN_GPIO, GPI);             // GPIO
pin, input mode, default pull
          // Lift Stepper Motor Setup
           liftPul.setup(LIFT_PUL_GPIO, GPO);                    // GPIO
pin, output mode, default pull
           liftDir.setup(LIFT_DIR_GPIO, GPO);                    // GPIO
pin, output mode, default pull
           liftEna.setup(LIFT_ENA_GPIO, GPO);                    // GPIO
pin, output mode, default pull
           liftDir.set(0);                                         //
Direction for lift motor
```

```
          liftEna.set(1);                                          // Lift
motor off
          liftTimer.setup(liftCallback, "lift_timer");
          // Variables
          int read;
          char msg[] = "Lifting mechanism...\nPlease wait...";
          liftEna.set(0); // Enable lift motor
          liftTimer.startPeriodic(3500); // Constant stepping speed
          lcdDisplay.printStr(msg);
          while(true) {
              read = heightSensor.get(); // Read height sensor
              if (read ==0) { // Height sensor triggered
                  liftTimer.stopPeriodic(); // Stop generating steps
                  liftEna.set(1); // Disable lift motor
                  lcdDisplay.printStr("Desired height\nreached!");
                  return true;
              }
                 vTaskDelay(pdMS_TO_TICKS(200)); // Wait ms before next
reading
          }
          return true;
      }

      bool tilting_motor() {
          // LCD Setup
          lcdDisplay.setup(lcd_pins);
          // Tilt stepper motor setup
          tiltPul.setup(TILT_PUL_GPIO, GPO);                       // GPIO
pin, output mode, default pull
          tiltDir.setup(TILT_DIR_GPIO, GPO);                       // GPIO
pin, output mode, default pull
          tiltEna.setup(TILT_ENA_GPIO, GPO);                       // GPIO
pin, output mode, default pull
           tiltDir.set(0);                                                //
Direction for tilt motor
          tiltEna.set(1);                                          // tilt
motor off
          // Timer setup
          tiltTimer.setup(tiltCallback, "tilt_timer");
          tiltStopTimer.setup(tiltStopCallback, "tilt_stop_timer");
          // Variables
          char msg[] = "Tilting basket...\nPlease wait...";
          lcdDisplay.printStr(msg);
```

```
          tiltEna.set(0); // Tilt motor ON
           tiltTimer.startPeriodic(15000);                                  //
Generate steps every 2 seconds
           tiltStopTimer.startOnce(4500000);                          // Stop
after 3 seconds
              vTaskDelay(pdMS_TO_TICKS(5500));  //  Wait  5.5  seconds  to
ensure the movement completes
          lcdDisplay.printStr("Tilting complete!");
          return true;
      }

      bool servomotor() {
          // Initialize
          lcdDisplay.setup(lcd_pins);
          servoMotor.setup(SERVOMOTOR_GPIO, 0);
          servoMotor.setDuty(0);
          // Actions
          lcdDisplay.printStr("Opening basket...\nUnloading beans...");
          servoMotor.setDuty(10);
          vTaskDelay(pdMS_TO_TICKS(1000));
          servoMotor.setDuty(0);
          lcdDisplay.printStr("Unloading\ncomplete!");
          return true;
      }

      extern "C" void app_main() {
          int next_state;
          bool good;
          states state = state0;
          for (int i = 0; i < 7; i++) {
              switch (state) {
                  case state0: // Setup all components
                      good = setup();
                      if (good == true) {
                          blinkLED(ledAct, 1, 1000);
                                                                  state   =
static_cast<states>(static_cast<int>(state) + 1);
                          continue;
                      }
                      else {
                          blinkLED(ledAct, 3);
                          exit(0);
                      }
```

```
                    break;
                case state1: //Move AGV without collision sensors
                    good = load_beans();
                    if (good == true) {
                        blinkLED(ledAct, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        continue;
                    }
                    else {
                        blinkLED(ledAct, 3);
                        exit(0);
                    }
                    break;
                case state2: //Move AGV without collision sensors
                    good = waiting_agv();
                    if (good == true) {
                        blinkLED(ledAct, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        continue;
                    }
                    else {
                        blinkLED(ledAct, 3);
                        exit(0);
                    }
                    break;
                case state3:
                    good = move_mechanism();
                    if (good == true) {
                        blinkLED(ledAct, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        continue;
                    }
                    else {
                        blinkLED(ledAct, 3);
                        exit(0);
                    }
                    break;
                case state4:
                    good = lifting_motor();
                    if (good == true) {
```

```
                        blinkLED(ledAct, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        continue;
                    }
                    else {
                        blinkLED(ledAct, 3);
                        exit(0);
                    }
                    break;
                case state5:
                    good = tilting_motor();
                    if (good == true) {
                        blinkLED(ledAct, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        continue;
                    }
                    else {
                        blinkLED(ledAct, 3);
                        exit(0);
                    }
                    break;
                case state6:
                    good = servomotor();
                    if (good == true) {
                        blinkLED(ledAct, 1, 1000);
                                                                state   =
static_cast<states>(static_cast<int>(state) + 1);
                        continue;
                    }
                    else {
                        blinkLED(ledAct, 3);
                        exit(0);
                    }
                    break;
            }
        }
    }
```

Annex (Scissor lift Definitions.h code)

```
#ifndef _DEFINITIONS_H_
#define _DEFINITIONS_H_

//Libraries to use
#include <SimpleADC.h>              //Analog signals
#include <SimpleGPIO.h>             //Digital signals
#include <SimpleKeypad.h>           //Keypad
#include <NibbleLCD.h>              //LCD
#include <SimplePWM.h>              //Motors
#include <SimpleTimer.h>            //Control time
#include <cmath>                    //Math functions

//GPIO pins

//  Basket servomotor
#define SERVOMOTOR_GPIO 36
//  Tilting stepper motor
#define TILT_PUL_GPIO 0
#define TILT_DIR_GPIO 32
#define TILT_ENA_GPIO 33
//  Lifting stepper motor
#define LIFT_PUL_GPIO 25
#define LIFT_DIR_GPIO 26
#define LIFT_ENA_GPIO 27
//  Height sensor
#define HEIGHT_SEN_GPIO 34
//  Load cell
#define LOAD_CELL_GPIO 39
//  Buzzer
#define BUZZER_GPIO 1
//  LCD
//                        D0  D1    D2    D3    D4    D5 D6  D7  RS  RW
EN
uint8_t  lcd_pins[11]  =  {13,  NULL,  NULL,  NULL,  NULL,  2,  16,  17,  14,
NULL, 12};
```

```
//  Keypad
uint8_t keypad_rows[4] = {5, 18, 19, 21};
uint8_t keypad_cols[4] = {15, 4, 22, 23};
//  Communication sensor
#define COMM_SENSOR_GPIO 35

//Object creation
//  Basket servomotor
SimplePWM servoMotor;
//  Tilting stepper motor
SimpleGPIO tiltPul; //Pulse
SimpleGPIO tiltDir; //Direction
SimpleGPIO tiltEna; //Enable
SimpleTimer tiltTimer;
SimpleTimer tiltStopTimer;
//  Lifting stepper motor
SimpleGPIO liftPul;
SimpleGPIO liftDir;
SimpleGPIO liftEna;
SimpleTimer liftTimer;
//  Height sensor
SimpleGPIO heightSensor;
//  Load Cell
SimpleADC loadCell;
// Buzzer
SimpleGPIO ledAct;
//  LCD
NibbleLCD lcdDisplay;
char lcdBuffer[100]; // Buffer for LCD display
//  Keypad
SimpleKeypad keypad(keypad_rows, keypad_cols);
//  Communication
SimpleGPIO slComSensor;

#endif // _DEFINITIONS_H_
```