Intro:

**Design Part:**

**Initial Plan**

* **Develop biomechanical wearable device – A device that moves the user whilst walking**
* **Research – Review papers for existing wearable devices**

**- Find out the biomechanics of turning**

* **Build designs – validate them and improve them**

**What we’ve done:**

* **Looked at review papers and found:**

**- Best feedback**

**- Devices that were discreet wearables using haptic feedback**

* **Found the biomechanics of turning:**

**- ...........**

* **Designed 6 devices**
* **Decided to focus on vest designs – focusing on upper body feedback**
* **Ranked designs using design specification – Top 2 were chosen based on the highest scores in the design specification table**

**Currently in the process of:**

* **Found common materials – ordered**
* **Estimate cost of designs**
* **Research possible ways of making the designs**
* **Learning how to use OpenSim to estimate forces required to move a person in desired direction**

**Next Step:**

* **Look at force reaction in opensim**
* **Choose design to move forward with**
* **Order materials**
* **Make device**
* **Test on ourselves with blindfold on**
* Biomechanics of turning
* Review papers results
* 6 wearable devices design
* Ranked using product design chart (Sam)

Next Step

Problems and solutions

* **Motor Driver Issues**:
* Problem: Single motor driver caused uneven current distribution, jerking motion, and loss of localization.
* Solution: Added a second motor driver and reconfigured Arduino code for smoother movement.
* **Frame Connectivity**:
* Issue: Odomtransformer node malfunction in the transform tree hierarchy.
* Solution: Troubleshot and fixed frame duplication conflicts, ensuring consistent odometry data.
* **Wireless Communication**:
* Problem: SSH was incompatible with eduroam Wi-Fi.
* Solution: Configured mobile hotspots for seamless communication.
* **ROS and Software Bugs**:
* Corrected typos in ROS repository URLs with custom search scripts.
* Updated legacy Python and ROS packages to match Noetic requirements.

### **Technical Workflow**

#### **Key Commands for Raspberry Pi & ROS:**

1. **Setup Workspace:**
   1. source catkin\_ws/devel/setup.bash
2. **Launch Lidar:**
   1. roslaunch ydlidar lidar.launch
3. **SLAM Mapping:**
   1. roslaunch hector\_slam\_launch tutorial.launch
4. **Teleoperation Control:**
   1. roslaunch ydlidar ydlidar\_teleop.launch
   2. rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py
5. **Localisation Integration:**
   1. roslaunch ydlidar localize2.launch

#### **System Optimisation:**

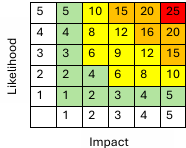
* Disable unused services for efficiency:
  + sudo systemctl disable cups (printer services)
  + sudo systemctl disable bluetooth (Bluetooth services)

#### **Next Steps in Testing:**

* Blindfolded trials for real-world feedback.
* Integration of OpenSim simulations into prototype refinement.

### **Summary**

* **Focus Areas:** Discreet, haptic wearable with upper-body feedback.
* **Achievements:** Research, design ranking, material selection, ROS integration.
* **Next Steps:** OpenSim simulations, material testing, and prototype manufacturing.

 Gantt chart link: [Gantt Chart.xlsx](https://cf.sharepoint.com/:x:/r/teams/MechatronicsWalk-Aid/Shared%20Documents/General/Gantt%20Chart.xlsx?d=w1b4d32a8f22f46c282126de048ac152a&csf=1&web=1&e=dxbqL3)

### **Challenges and Solutions**

**Challenge 1: Software Compatibility Issues**

* **Details:** Initially attempted to use the more common RPLidar library, but this was incompatible with our YDLidar sensor. Transitioned to the YDLidar library, which required troubleshooting and adapting legacy code.
* **Solution:** Researched and implemented an older YDLidar GitHub package, resolving compatibility issues despite the initial setup hurdles.

**Challenge 2: SSH Connectivity**

* **Details:** Attempted to use the Raspberry Pi’s hotspot for SSH but encountered failures. The University’s WiFi network also proved incompatible.
* **Solution:** Configured SSH using our mobile hotspots. Later acquired a modem for consistent access.

**Challenge 3: USB Port Aliasing for Lidar**

* **Details:** The Raspberry Pi initially failed to recognize the USB port linked to the lidar.
* **Solution:** Used troubleshooting guidance to manually alias the USB port, ensuring proper communication with the YDLidar sensor.

**Challenge 4: ROS Installation Issues**

* **Details:** Encountered errors with missing libraries and incorrect configurations when installing RViz on the Raspberry Pi. Discovered that one of the repository links in the ROS source list contained a typo (htp instead of http).
* **Solution:** Created and ran a script to search all source files for similar typos. Corrected the faulty URLs, enabling successful RViz installation and Hector SLAM integration.

**Challenge 5: Pin Assignment Conflicts**

* **Details:** Conflicts arose when certain digital pins required for motor drivers were also allocated for Raspberry Pi communication.
* **Solution:** Reassigned pin configurations, creatively utilizing analog pins as digital pins where possible.

**Challenge 6: Motor Control and Localisation Stability**

* **Details:** Initial motor control caused abrupt, jerky movements, disrupting localisation during mapping. Additionally, a single motor driver led to uneven power distribution.
* **Solution:** Replaced the single motor driver with two drivers for balanced control. Fine-tuned Arduino motor code to ensure smoother movements, which improved map quality.

**Challenge 7: Legacy Python and ROS Compatibility**

* **Details:** Many ROS packages and scripts, such as odomtransformer, were designed for Python 2, while our setup required Python 3 for compatibility with ROS Noetic.
* **Solution:** Updated and converted outdated Python 2 code to Python 3 syntax, ensuring smooth functionality across all nodes.

**Challenge 8: Hector SLAM Performance**

* **Details:** Fast motor speeds led to frequent localisation errors. Additionally, TF tree misconfigurations caused delays in the transform hierarchy.
* **Solution:** Minimized robot movement during mapping sessions to reduce localisation errors. Reconfigured the TF tree hierarchy to resolve transform issues.

**Challenge 9: Battery Management**

* **Details:** The vehicle's battery depleted unexpectedly, halting testing.
* **Solution:** Coordinated with the University to recharge the battery and considered investing in a secondary power source for uninterrupted progress.

### **Sample Examiner Questions and Answers**

#### **General Questions**

**Q: How is your device different from existing solutions?**

A: Our device focuses on upper-body haptic feedback, which is discreet and effective. Unlike canes or large wearables, our vest design integrates seamlessly with everyday clothing while providing precise navigation cues.

**Q: Why did you use ROS?**

A: ROS simplifies sensor integration, SLAM, and communication between components. Its modularity enables efficient prototyping and scalability for future improvements.

#### **Technical Questions**

**Q: How does SLAM work in your system?**

A: SLAM uses the Lidar sensor to map the environment in real-time. The Hector SLAM package processes Lidar data to create a 2D map, allowing the device to navigate and localize effectively.

**Q: How do you prevent interference between sensors and actuators?**

A: We isolated power supplies using voltage regulators and ensured separate communication channels for sensors (I2C for Lidar) and actuators (PWM for motors).

#### **Design Questions**

**Q: What materials are used in your vest design?**

A: Lightweight, breathable fabrics for comfort, combined with haptic actuators (e.g., vibration motors) placed strategically for clear feedback.

**Q: How do you ensure the vest fits various users?**

A: The vest features adjustable straps and modular feedback modules that can be repositioned based on user preferences.

#### **Testing Questions**

**Q: How will you ensure your device is safe for users?**

A: We will conduct force testing in OpenSim to ensure feedback is within safe limits. Additionally, blindfolded trials will be supervised to minimize risks.

**Q: What are your testing criteria?**

A: Navigation accuracy, user comfort, response time, and device discreteness.

### **Why did you choose Hector SLAM over other SLAM algorithms?**

**A**: Hector SLAM is highly optimized for LiDAR-based mapping and offers robust performance on low-power platforms like the Raspberry Pi. Its reliance on scan matching rather than odometry makes it ideal for environments where wheel slip or sensor drift might occur.

### **How did you address motor synchronization issues?**

**A**: We identified current imbalance due to a single motor driver. Adding a second motor driver allowed equal current distribution, eliminating jerking and improving localization accuracy. The Arduino code was updated accordingly.

### **What measures did you take to optimise computational efficiency?**

**A**: We also tuned LiDAR, navigation and localisation parameters and minimized redundant node communication. Ie all the control loops attempt to operate at the same frequency

### **How did you troubleshoot the odomtransformer node?**

**A**: We identified frame conflicts in the transform tree hierarchy. By eliminating duplicate frames and ensuring proper topic synchronisation, the node performed as expected.

### **What considerations were made for user comfort in the design?**

**A**: We focused on creating a lightweight, wearable device that integrates seamlessly with natural movements. Feedback methods like vibration and guiding weight were explored to ensure intuitive interaction.

### **What challenges did you face during integration, and how did you resolve them?**

**A**: The main challenges included communication setup (e.g., SSH issues), frame conflicts, and motor synchronization. Incremental testing, scripting fixes for typos, and rigorous troubleshooting resolved these issues.