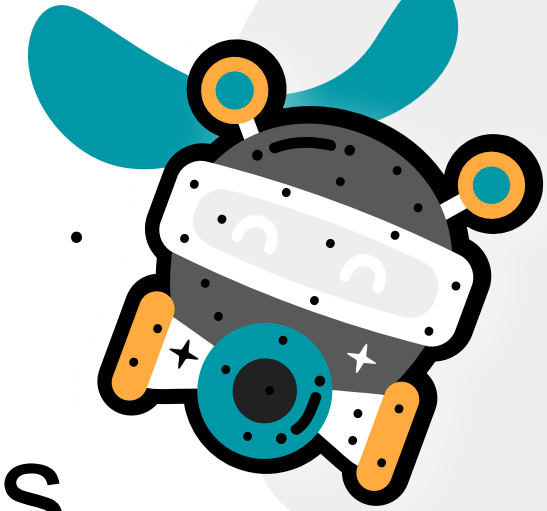
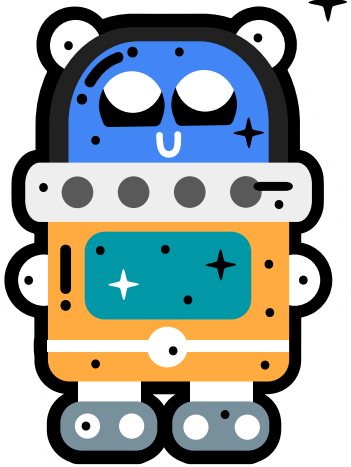


Stretch RE1 Voice Controls



Team 3

Apple Zhao, Kevin Chou, Raj Mehta

Problem Definition

Implementing voice teleoperation functionality on a stretch robot to give a user the ability to manipulate all its joints and complete dexterous tasks

Challenges

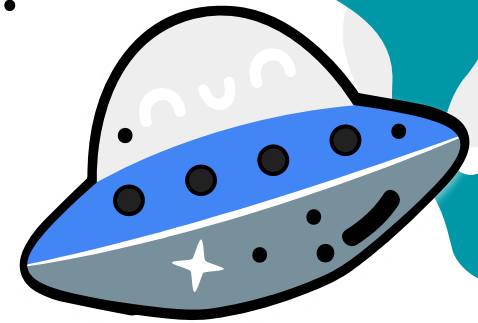
Converting voice commands to actions

Giving a user ability to fully manipulate the robot

Assumption

Assuming users are able to speak and have spatial understanding

Why Voice Control?



- No physical controller → removes dexterity barrier
- No graphical user interface → lowers technical barrier
- No extra hardware required → lowers technical barrier
- More intuitive → lowers technical barrier
- Accessible to anyone that can speak – e.g. easily fatigued individuals
- Specific Target Population: Spinal cord injury patients – 291,000 people in the U.S, 60% of whom are quadraplegic

Why Teleoperation?

- **Teleoperation in Literature and in the real World:**
  - Poncela et. al. used voice commands to successfully teleoperate a Pioneer P2AT robot for navigation tasks
  - Lu et al. used voice operation to control a humanoid robot, moves joints and performs actions (walk, play, etc.)
  - Amazon Alexa: Can use voice commands to set reminders, turn on/off utilities; Amazon Astrobot works on voice teleoperation to navigate a home environment

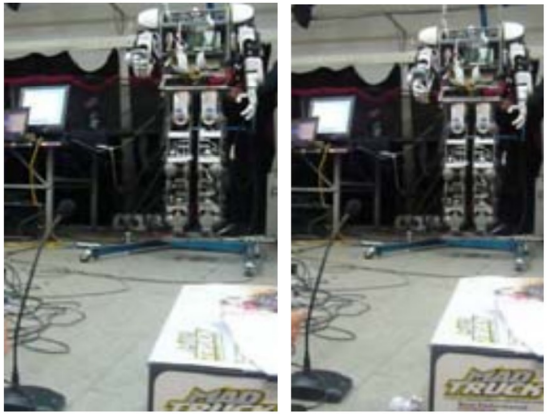


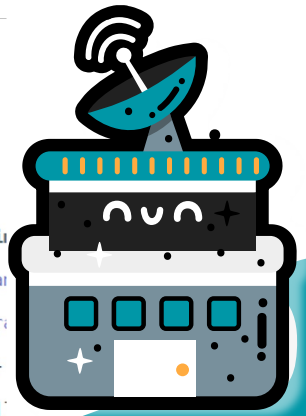
Figure.8 Voice teleoperation for humanoid arm

Implementation

Based on HelloRobot's "Voice Teleoperation of Base" Example

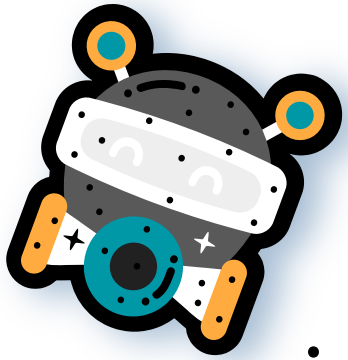
```
def get_inc(self):
    """
    A function that sets the increment size for translational and rotational
    base motion.
    :param self:The self reference.

    :returns inc: A dictionary type the contains the increment size.
    """
    translation = self.medium_translate
    rotation = self.medium_rad
    aperture = self.aperture
    if self.command_list:
        for s in self.command_list:
            if s.isnumeric():
                translation = int(s)

    translation = translation/100
    if 'meter' in self.command_list
        translation = translation*10
```

```
def get_command(self):
    """
    A function that defines the teleoperation command based on the voice command.
    :param self: The self reference.

    :returns command: A dictionary type that contains the type of base motion.
    """
    command = None
    if self.voice_command and self.command_list:
        if ('base' in self.command_list) or ('face' in self.command_list) or ('space' in
            if ('forward' in self.command_list) or ('Forward' in self.command_list):
                command = {'joint': 'translate_mobile_base', 'inc': self.get_inc()['tra
            if 'back' in self.command_list:
                command = {'joint': 'translate_mobile_base', 'inc': -self.get_inc()['tra
            if 'left' in self.command_list:
                command = {'joint': 'rotate_mobile_base', 'inc': self.get_inc()['rad']}
            if 'right' in self.command_list:
                command = {'joint': 'rotate_mobile_base', 'inc': -self.get_inc()['rad']
```



Implementation

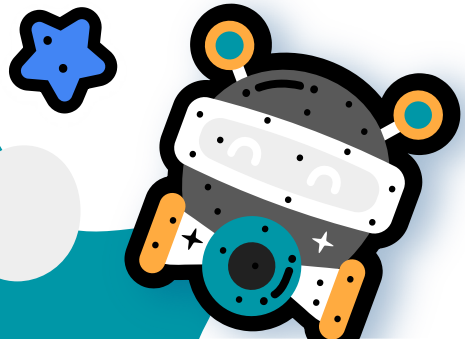
```
def send_command(self, command):
    """
    Function that makes an action call and sends base joint trajectory goals
    :param self: The self reference.
    :param command: A dictionary type.
    """
    joint_state = self.joint_state
    if (joint_state is not None) and (command is not None):

        inc = command['inc']
        rospy.loginfo('inc = {0}'.format(inc))
        new_value = inc
        joint_name = command['joint']

        if joint_name == 'translate_mobile_base':
            pose = {'translate_mobile_base': new_value}
            self.move_to_pose(pose)
            rospy.sleep(1.0)
```



Build Your Own Commands (Midterm)

Choose a joint:

- Base
- Lift
- Arm
- Base
- Wrist
- Grip



Choose a direction:

Base:
- Forward
- Back
- Left
- Right

Lift:
- Up
- Down

Arm:
- Extend
- Retract

Wrist (absolute):
- Up
- Down
- Left
- Right
- Roll

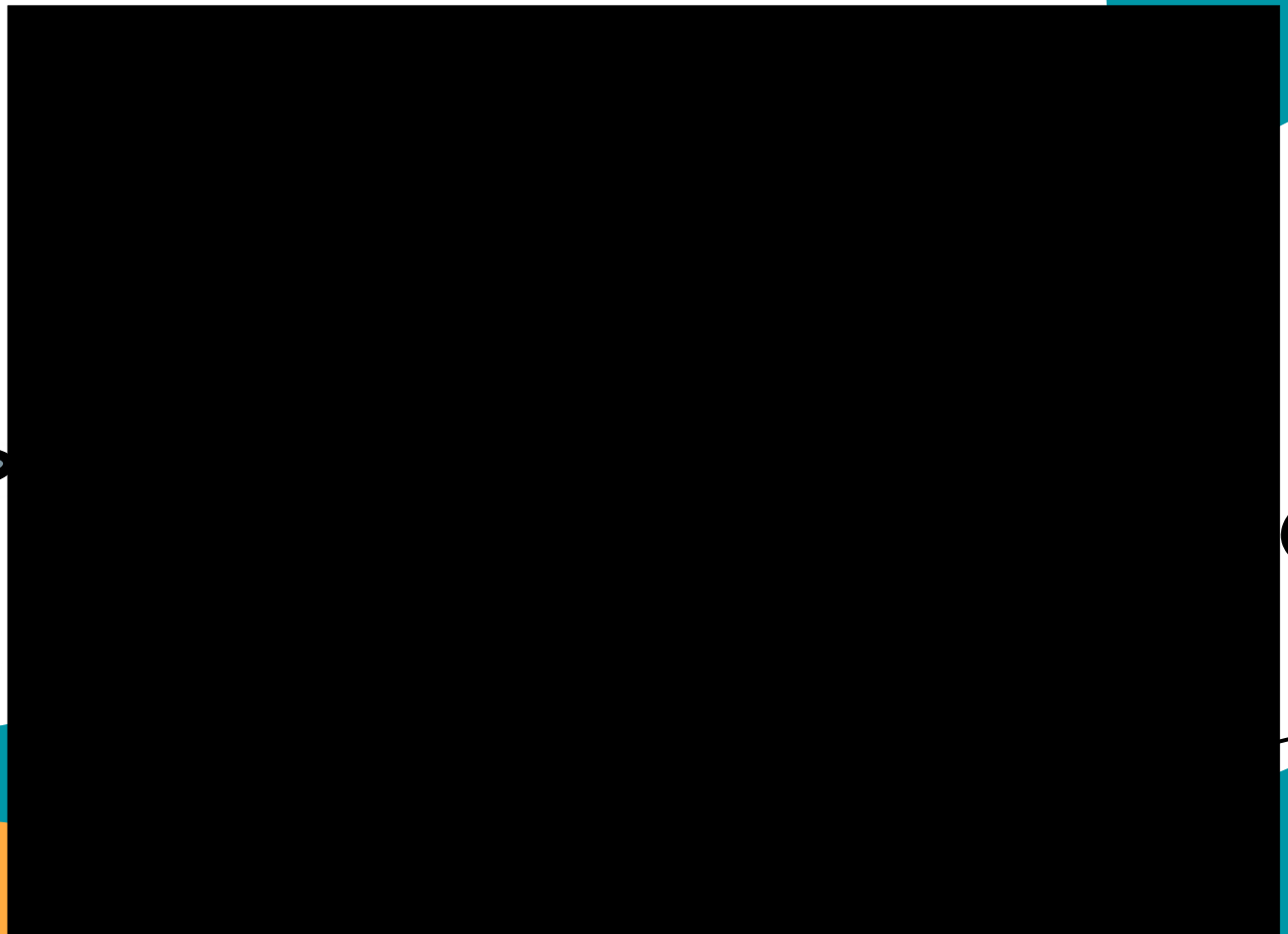
Grip:
- Open
- Close

Choose an increment:

A number
- 10
- 15
- 20
- ...

Units
- Meters, cm, degrees

Features in Midterm (2x)



Midterm Stakeholder Feedback

Dr. William Mills

William Mills
to me

Hi Kevin,
This is a great project. I certainly agree that there is value in designing a set of voice commands for the robot to help patients who are unable to use a controlled for one reason or another. One thing to keep in mind is that in a case where precision is needed, it is difficult for most people to closely estimate distance and angles. Maybe there could be a way to measure first? Or to somehow superimpose a measurement matrix on the object and person for more precise measurement? In other words, I worry that people may be trying to get the robot to interact with something a few feet away, but they don't know the combination of distance and angles to use in a voice command to do the task?

Henry Evans

Henry Evans
to me

demo looks good , but consider this :

Most Americans, ESPECIALLY the elderly A] aren't familiar with the metric system, and B] aren't particularly good at judging degrees or units of distance needed.
As an optional alternative , consider a 'START/STOP' approach, where you first say 'start rotating wrist to the left' and, when it approaches the position you want, say 'STOP!'. For safety, automatically stop all motion 5 seconds after it starts.

Another feature would be to verbally initiate pre-recorded motion sequences like 'flip the lightswitch ON'

Features We Added

Pose Saving

The user can save joint positions for repetitive tasks

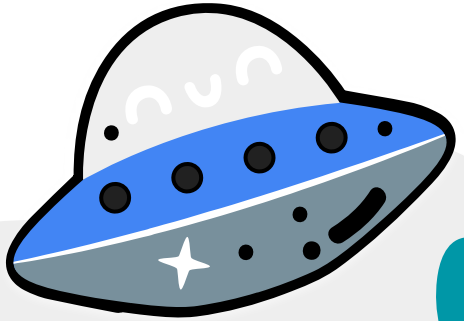
Start/Stop Control

More fine control of joints through minor increments

Improved Teleoperation

Can operate the robot from anywhere in the world!

*Also added imperial unit functionality

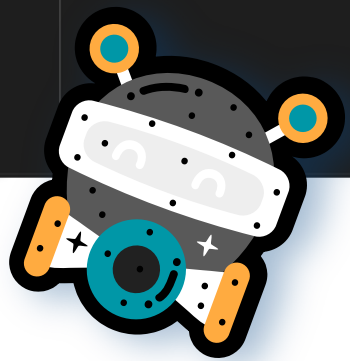


Implementation

```python
if 'save' in command:
    joint_state_dict = dict()
    for idx, elem in enumerate(joint_state.name):
        joint_state_dict[elem] = joint_state.position[idx]

    json_object = json.dumps(joint_state_dict)
    # Writing to sample.json
    name = command['save']
    with open(f'poses/{name}.json', "w") as outfile:
        outfile.write(json_object)

if 'run' in command:
    filename = command['run']

    file_list = os.listdir('poses/')

    file_list = list(map(lambda x:x.split('.')[0], file_list))

    if filename in file_list:
        with open(f'poses/{filename}.json') as json_file:
            pose = json.load(json_file)
            print(pose)
            for key in pose:
                new_pose = {key: pose[key]}
                print(new_pose)
                self.move_to_pose(new_pose)
```

```python
while not rospy.is_shutdown():
    command = self.speech.get_command()
    if self.speech.keep_moving_flag:
        command = {'joint': self.speech.keep_moving_joint}
        if self.speech.keep_moving_joint == 'joint_lift':
            command['inc'] = -0.03 if self.speech.inc_negative == False else 0.03
        elif self.speech.keep_moving_joint == 'translate_mobile_base' or self.speech.keep_moving_joint == 'rotate_mobile_base':
            command['inc'] = 0.1 if self.speech.inc_negative == False else -0.1
        elif self.speech.keep_moving_joint == 'wrist_extension':
            command['inc'] = 0.05 if self.speech.inc_negative == False else -0.05
    self.send_command(command)
    rate.sleep()
```

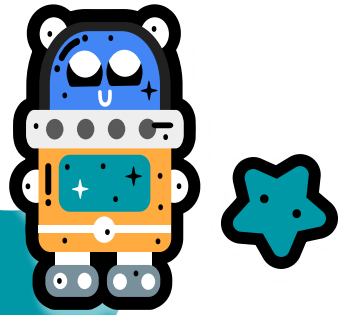
```python
if (self.voice_command and self.command_list) or self.keep_moving_flag:
    if self.keep_moving_flag and (self.command_list is not None) and ("stop" in self.command_list):
        command = {'joint': self.keep_moving_joint, 'inc': 0}
        self.keep_moving_flag = False
        self.voice_command = None
        self.command_list = None
        self.inc_negative = False
        print("I heard stop \n \n \n \n \n \n")
        return command
```



Build Your Own Commands (Current)

Special Command:

- Keep Moving
  + (Joint & Direction)
- Save
  [/name_of_pose]
- Run
  [/saved_name_of_pose]



Choose a joint:

- Base
- Lift
- Arm
- Base
- Wrist
- Grip

Choose a direction:

Base:
- Forward
- Back
- Left
- Right

Lift:
- Up
- Down

Arm:
- Extend
- Retract

Wrist (absolute):
- Up
- Down
- Left
- Right
- Roll

Grip:
- Open
- Close

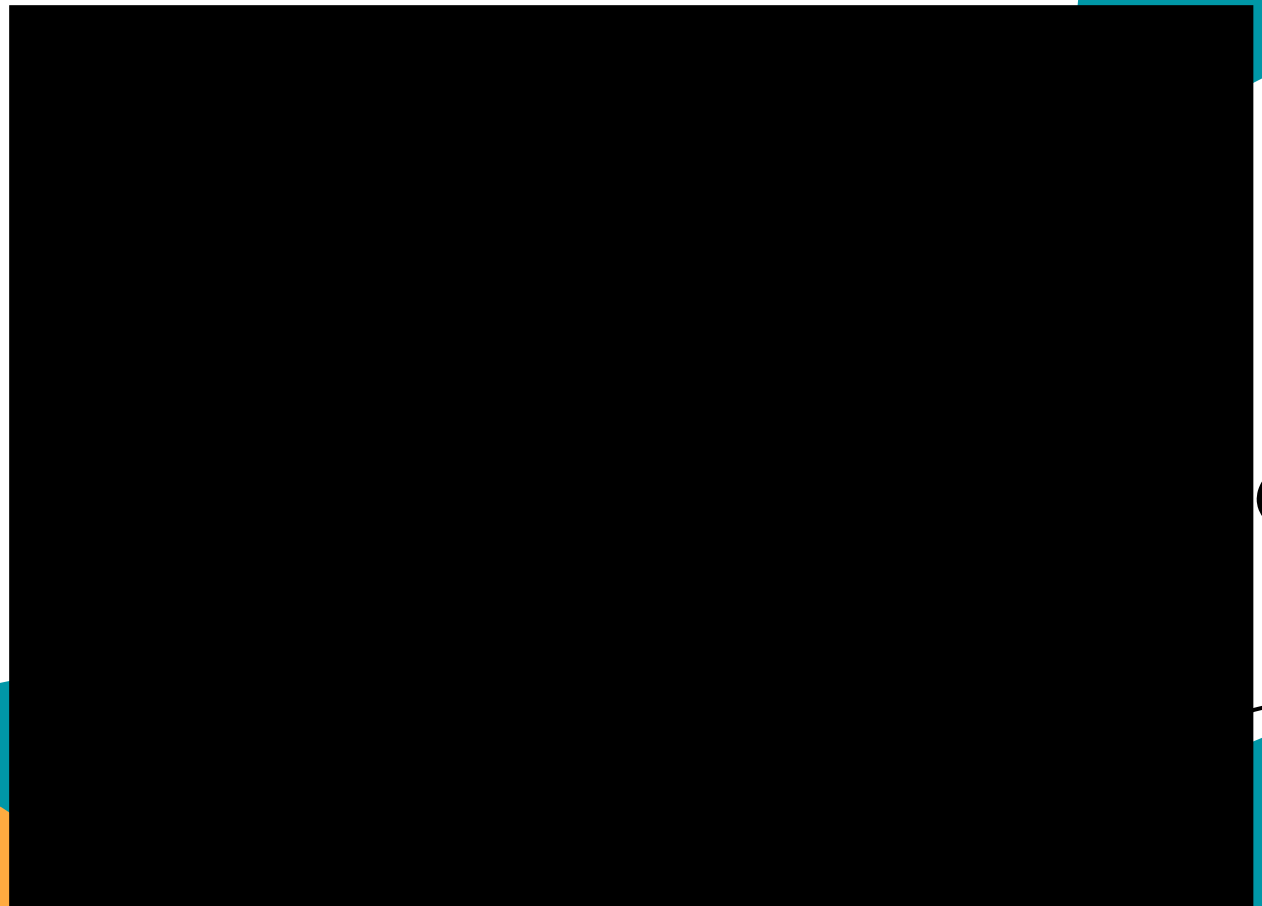
Choose an increment:

A number
- 10
- 15
- 20
- …

Units
- Meters, cm, degrees, inches, feet

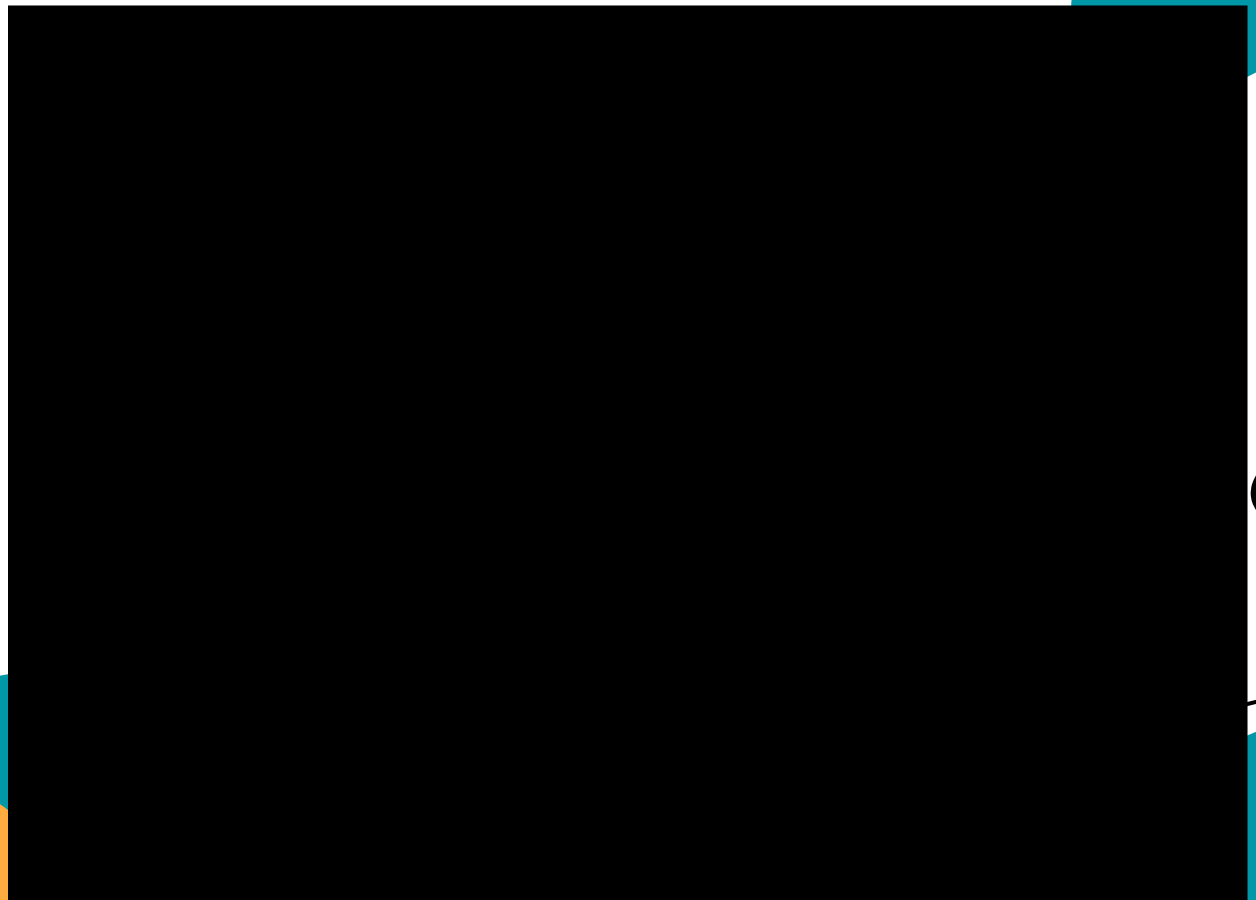
Pose Saving

Start/Stop



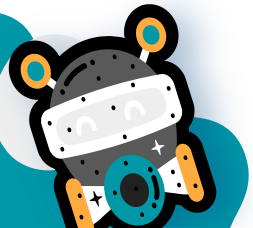
Evaluation 1 (4x Speed)

Evaluation 2 (2.5x Speed)



Evaluation Results

- Teleoperation requires high spatial understanding
- Total time to complete task was ~3 minutes
- Through pose saving subsequent tasks can be completed faster (1:30)
- Keep moving and save/run pose commands greatly reduce time and effort



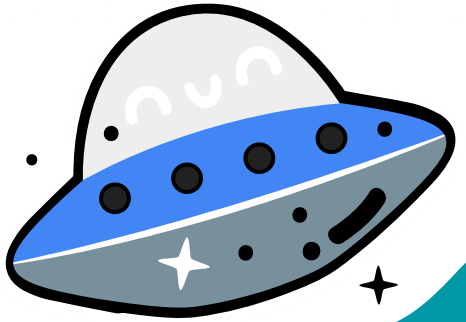
Stakeholder Interaction with Changes!

Henry Evans

Henry Evans
to me

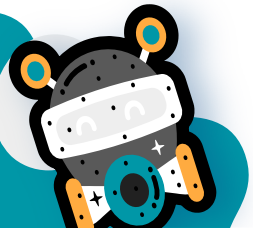
That's very impressive. Couple thoughts:
-your 'stop' command is not nearly instantaneous; it would, as currently configured, be difficult to stop it exactly where you wanted
-the prerecorded movement sequence is really cool; now try recording one that does something useful;eg;'scratch my cheek/head ', etc. etc. ...
Cheers,

Henry Evans



Next Steps

- **Run Voice Teleop on startup**
- **Bluetooth earphone support**
  - Currently implemented through a rudimentary approach - calling a mobile phone
- **Useful saved poses**
- **Fix the start-stop lagging gap**



References

- Poncela, A., & Gallardo-Estrella, L. (2015). Command-based voice teleoperation of a mobile robot via a human-robot interface. *Robotica, 33*(1), 1-18. doi:10.1017/S0263574714000010

- Y. Lu, L. Liu, S. Chen and Q. Huang, "Voice Based Control for Humanoid Teleoperation," 2010 International Conference on Intelligent System Design and Engineering Application, Changsha, China, 2010, pp. 814-818, doi: 10.1109/ISDEA.2010.430.

Thank you for listening!

Any Questions? :D