Conntact
Tactile Assembly Framework

SOUTHWEST RESEARCH INSTITUTE®

https://github.com/swri-robotics/ConnTact
Goals

**Agile** Easy to set up, modify, and repurpose

**Hardware-Agnostic** Algorithms function without modification on different computers and robots

https://github.com/swri-robotics/ConnTact
Compliant Robotics

UR10e running Cartesian Compliance Controller
https://github.com/fzi-forschungszentrum-informatik/cartesian_controllers
Spiral Search Peg Insertion


https://github.com/swri-robotics/ConnTact
class SpiralSearch(ConnTask):

    def __init__(self, context, interface, config_name):

        # Declare the official states list here. These will be passed into the machine.
        states = [
            START_STATE,
            APPROACH_STATE,
            FIND_HOLE_STATE,
            INSERTING_PEG_STATE,
            COMPLETION_STATE,
            EXIT_STATE,
            SAFETY_RETRACT_STATE
        ]

        # Define the valid transitions from/to each state. Here's where you define the topology of the state machine.
        # The Machine executes the first transition in this list which matches BOTH the trigger AND the CURRENT state.
        # If no other trigger is set at "self.next_trigger", Conntact will automatically fill in "RUN_LOOP_TRIGGER"
        # which runs the Execute method of the current Step object.
        transitions = [
            {'trigger': APPROACH_SURFACE_TRIGGER, 'source': START_STATE, 'dest': APPROACH_STATE},
            {'trigger': STEP_COMPLETE_TRIGGER, 'source': FIND_HOLE_STATE, 'dest': FIND_HOLE_STATE},
            {'trigger': STEP_COMPLETE_TRIGGER, 'source': INSERTING_PEG_STATE, 'dest': INSERTING_PEG_STATE},
            {'trigger': STEP_COMPLETE_TRIGGER, 'source': SAFETY_RETRACT_STATE, 'dest': SAFETY_RETRACT_STATE,
             unless: 'is_already_retracting'},
            {'trigger': Step_COMPLETE_TRIGGER, 'source': SAFETY_RETRACT_STATE, 'dest': APPROACH_STATE},
            {'trigger': RUN_LOOP_TRIGGER, 'source': SAFETY_RETRACT_STATE, 'dest': None, after: run_step_actions}
        ]

        self.step_list = {APPROACH_STATE: (FindSurface, []),
                          FIND_HOLE_STATE: (SpiralToFindHole, []),
                          INSERTING_PEG_STATE: (FindSurfaceFullCompliant, []),
                          SAFETY_RETRACT_STATE: (SafetyRetraction, []),
                          COMPLETION_STATE: (ExitStep, [])}

        # # Initialize the state machine "Machine" init in your Conntask instance
        ConnTask.__init__(self, context, states, transitions, config_name=config_name)
SpiralSearch Code Solution: State Behaviors

“Move down until you bump into something, and record the surface height”

“Move outward in a spiral until you drop past the surface”

(Math to define a spiral)

```python
class FindSurface(ConnStep):
    def __init__(self, connTask: ConnTask) -> None:
        ConnStep.__init__(self, connTask)
        self.comoly_axes = [0, 0, 1]
        self.seeking_force = [0, 0, -7]

    def exit_conditions(self) -> bool:
        return self.is_static() and self.in_collision()

    def on_exit(self):
        # Measure flat surface height and report it to AssemblyBlocks:
        self.connTask.surface_height = self.connTask.current_pose.transform.translation.z
        return super().on_exit()
```

```python
class SpiralToFindHole(ConnStep):
    def __init__(self, connTask: ConnTask) -> None:
        ConnStep.__init__(self, connTask)
        self.seeking_force = [0, 0, -7]
        self.spiral_params = self.connTask.config['spiral_params']
        self.safe_clearance = self.connTask.config['objects']['dimensions']['safe_clearance'] / 100
        self.start_time = self.connTask.interface.get_unified_time()

    def update_commands(self):
        # Updates the commanded position and wrench. These are published in the ConnTask main loop.
        ...
        # Command wrench
        self.connTask.wrench_command_vector = self.connTask.get_command_wrench(self.seeking_force)
        # Command pose
        self.connTask.pose_command_vector = self.connTask.get_spiral_search_pose()

    def exit_conditions(self) -> bool:
        return self.connTask.current_pose.transform.translation.z == self.connTask.surface_height - .0084
```

https://github.com/swri-robotics/ConnTact
SpiralSearch Code Solution: ROS node

```python
conntasks = {
    "SpiralSearch": SpiralSearch
}

if __name__ == '__main__':
    rospy.init_node("demo_assembly_application_compliance")

    interface = ConntactROSInterface("conntact_params")
    conntext = Conntext(interface)

    params = interface.load_yaml_file("peg_in_hole_params")
    task_info = params["conntact_info"]['task_list']
    interface.register_frames(read_board_positions(task_info['position'], params))

    # The below could be run in a loop to execute all tasks specified in the task_list. Not currently implemented.

    # Instantiate the task called for in the task_list:
    task = conntasks[task_info['task']](conntext, interface, task_info["conntFig"])

    # ** Here's where you would do pathing stuff to move the robot close to the task location.
    # Begin the Task:
    task.main()

    interface.send_info(Style.MAGENTA + "Node has control again!" + Style.RESET_ALL)
```

All Tasks needed for this application

Instantiate Interface and Conntext

Instantiate the Conntask, passing in ConntFig, then run

https://github.com/swri-robotics/ConnTact
Example program summary:
• 230 lines of code
• 2 YAML files

Upcoming goals:
• Full ROS 2 support
• More example applications