# *Conntact*
# Tactile Assembly Framework

SOUTHWEST RESEARCH INSTITUTE®

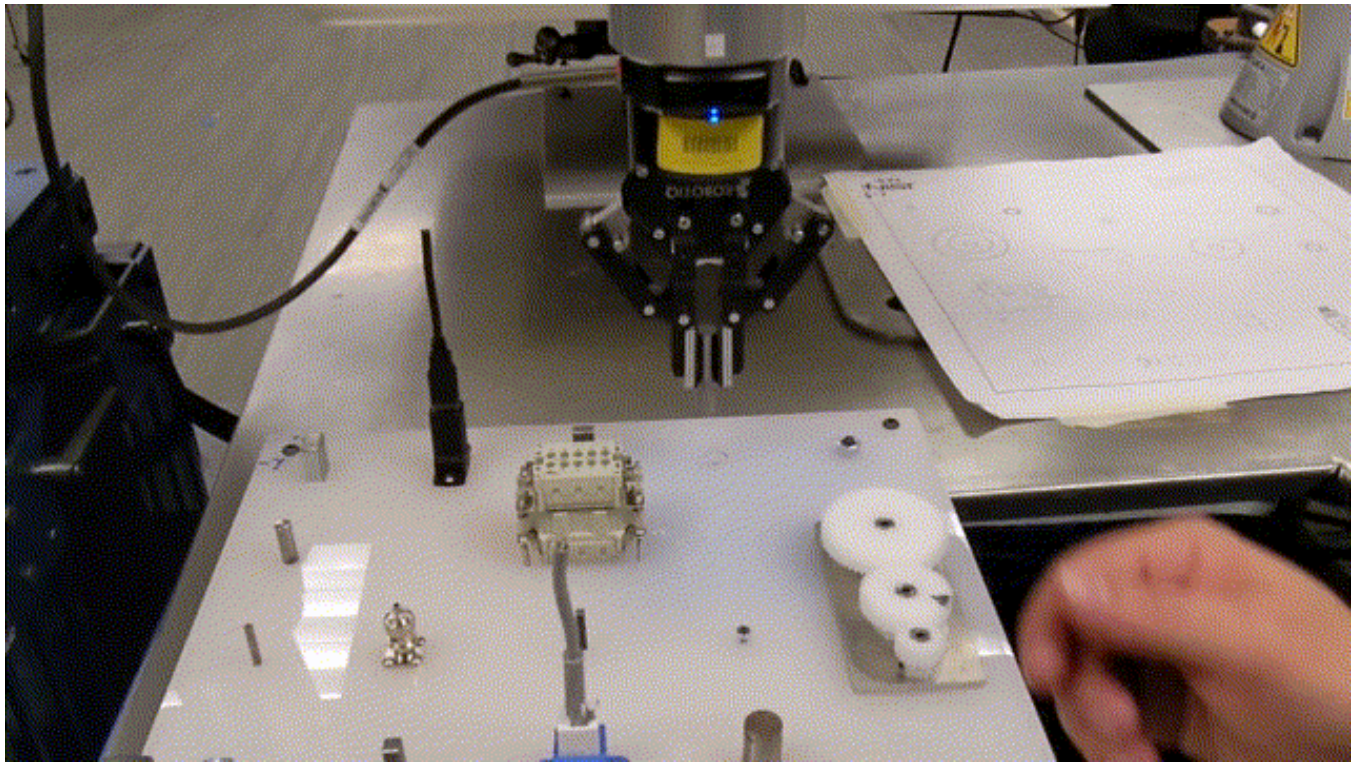
https://github.com/swri-robotics/ConnTact

# Goals

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

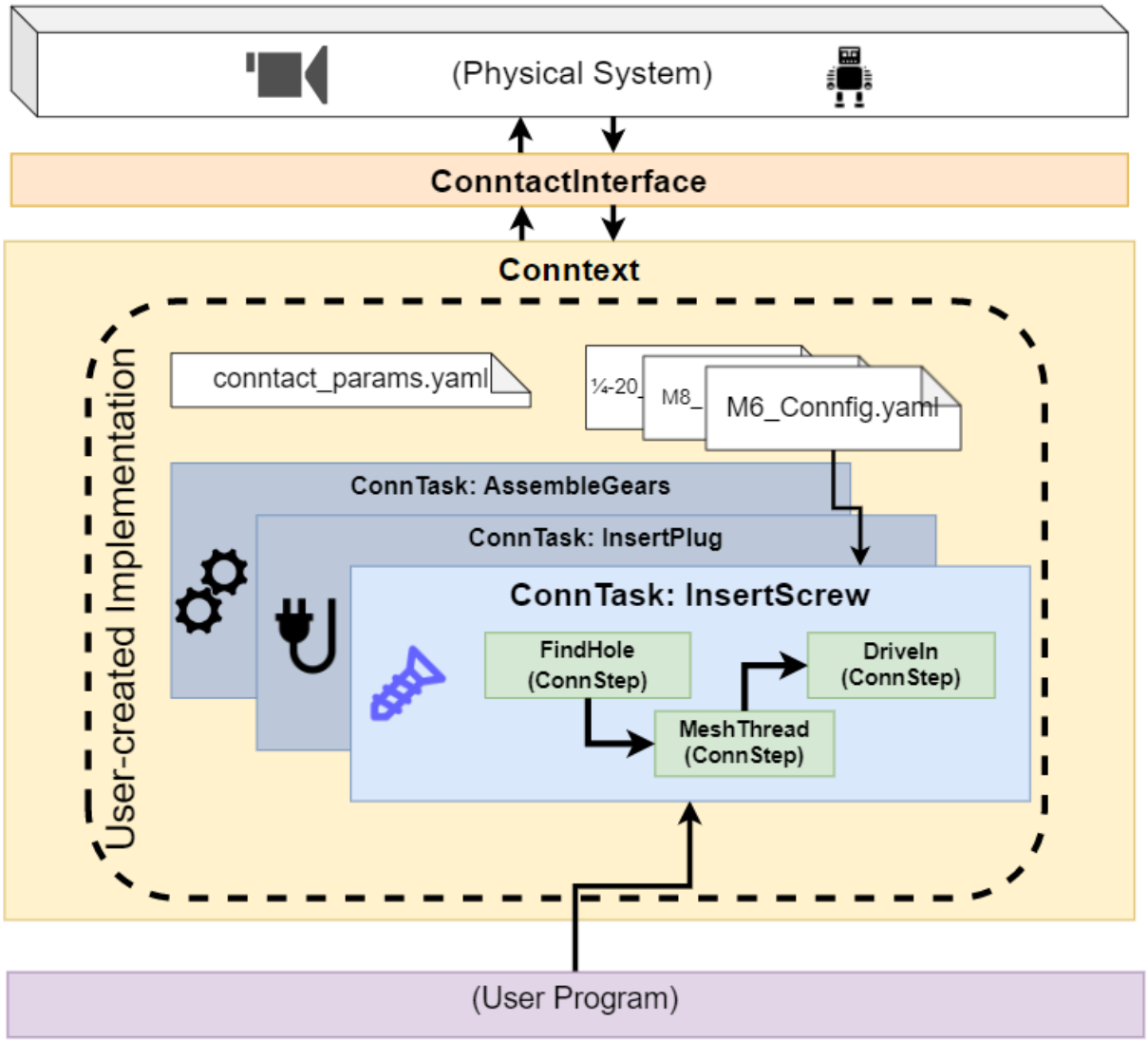
*Hardware-Agnostic* — Algorithms function without modification on different computers and robots

# Compliant Robotics

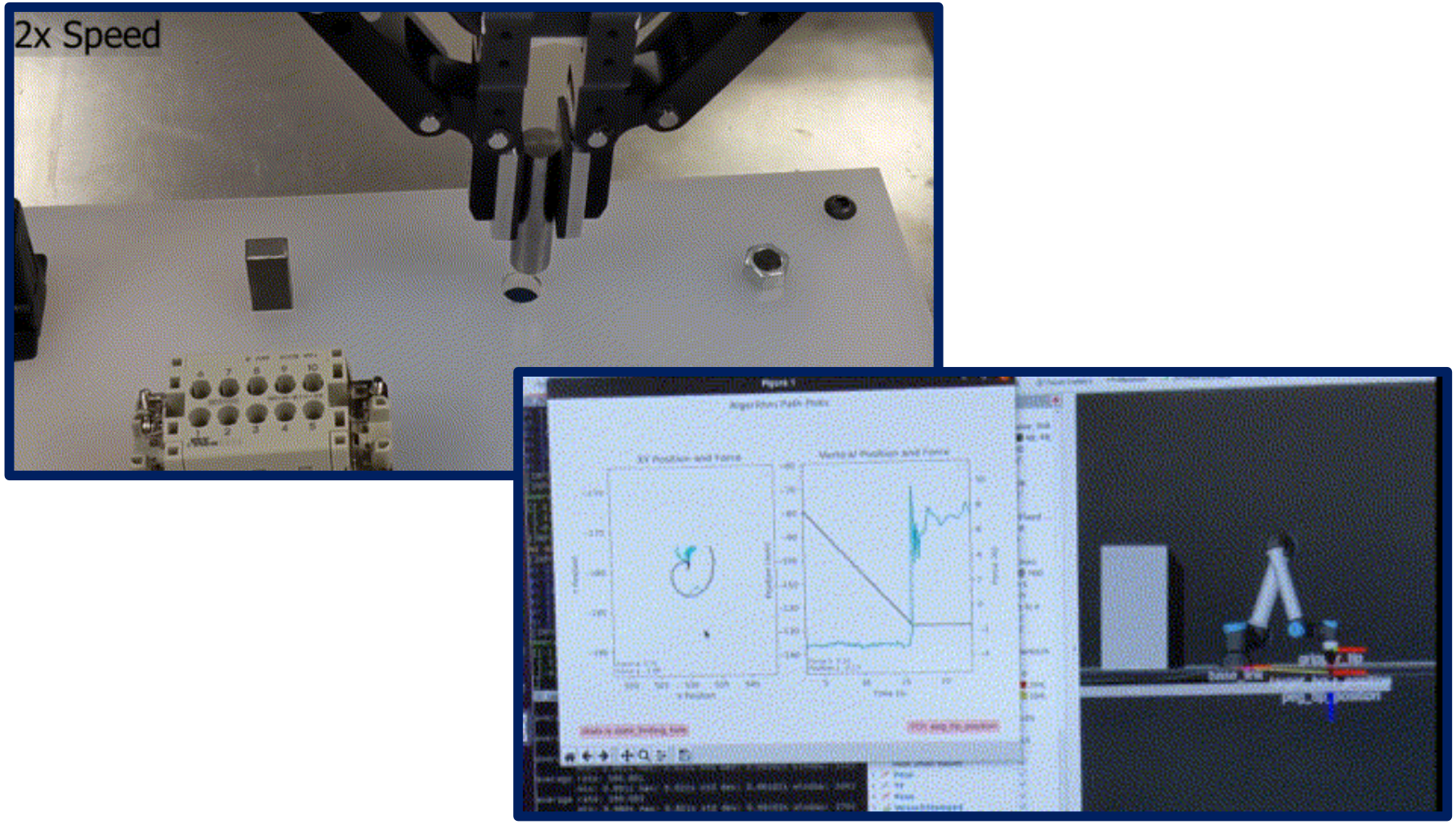


UR10e running Cartesian Compliance Controller
https://github.com/fzi-forschungszentrum-informatik/cartesian_controllers

# Overview



# Spiral Search Peg Insertion




https://rosindustrial.org/news/2021/10/14/introducing-the-conntact-assembly-framework

# SpiralSearch Code Solution: State Machine

Declare states

Declare the transitions between states

Attach a behavior to each state

```
class SpiralSearch(ConnTask):

    def __init__(self, conntext, interface, connfig_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':APPROACH_STATE      , 'dest':FIND_HOLE_STATE        },
            {'trigger':STEP_COMPLETE_TRIGGER     , 'source':FIND_HOLE_STATE     , 'dest':INSERTING_PEG_STATE    },
            {'trigger':STEP_COMPLETE_TRIGGER     , 'source':INSERTING_PEG_STATE , 'dest':COMPLETION_STATE       },
            {'trigger':STEP_COMPLETE_TRIGGER     , 'source':COMPLETION_STATE    , 'dest':EXIT_STATE             },
            {'trigger':SAFETY_RETRACTION_TRIGGER , 'source':'*'                 , 'dest':SAFETY_RETRACT_STATE,
              'unless':'is_already_retracting' },
            {'trigger':STEP_COMPLETE_TRIGGER     , 'source':SAFETY_RETRACT_STATE, 'dest':APPROACH_STATE         },
            {'trigger':RUN_LOOP_TRIGGER          , 'source':'*'                 , 'dest':None, 'after':
'run_step_actions'}
        ]

        self.step_list:dict = { 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, conntext, states, transitions, connfig_name=connfig_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.comply_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):
        """Executed once, when the change-state trigger is registered.
        """
        # Measure flat surface height and report it to AssemblyBlocks:
        self.task.surface_height = self.conntext.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.task.connfig['task']['spiral_params']
        self.safe_clearance = self.task.connfig['objects']['dimensions']['safe_clearance']/100 #convert to m
        self.start_time = self.conntext.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.task.wrench_command_vector  = self.conntext.get_command_wrench(self.seeking_force)
        #Command pose
        self.task.pose_command_vector = self.get_spiral_search_pose()

    def exit_conditions(self) -> bool:
        return self.conntext.current_pose.transform.translation.z <= self.task.surface_height - .0004

    def get_spiral_search_pose(self):
        """Generates position, orientation offset vectors which describe a plane spiral about z;
        Adds this offset to the current approach vector to create a searching pattern. Constants come from Init;
        x,y vector currently comes from x_ and y_pos_offset variables.
        """
        # frequency=.15, min_amplitude=.002, max_cycles=62.83185
        curr_time = self.conntext.interface.get_unified_time() - self.start_time
        curr_time_numpy = np.double(curr_time.to_sec())
        frequency = self.spiral_params['frequency'] #because we refer to it a lot
        curr_amp = self.spiral_params['min_amplitude'] + self.safe_clearance * \
                   np.mod(2.0 * np.pi * frequency * curr_time_numpy, self.spiral_params['max_cycles']);
        x_pos = curr_amp * np.cos(2.0 * np.pi * frequency * curr_time_numpy)
        y_pos = curr_amp * np.sin(2.0 * np.pi * frequency * curr_time_numpy)
        x_pos = x_pos + self.task.x_pos_offset
        y_pos = y_pos + self.task.y_pos_offset
        z_pos = self.conntext.current_pose.transform.translation.z
        pose_position = [x_pos, y_pos, z_pos]
        pose_orientation = [0, 1, 0, 0]  # w, x, y, z

        return [pose_position, pose_orientation]
```

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

# SpiralSearch Code Solution: ROS node

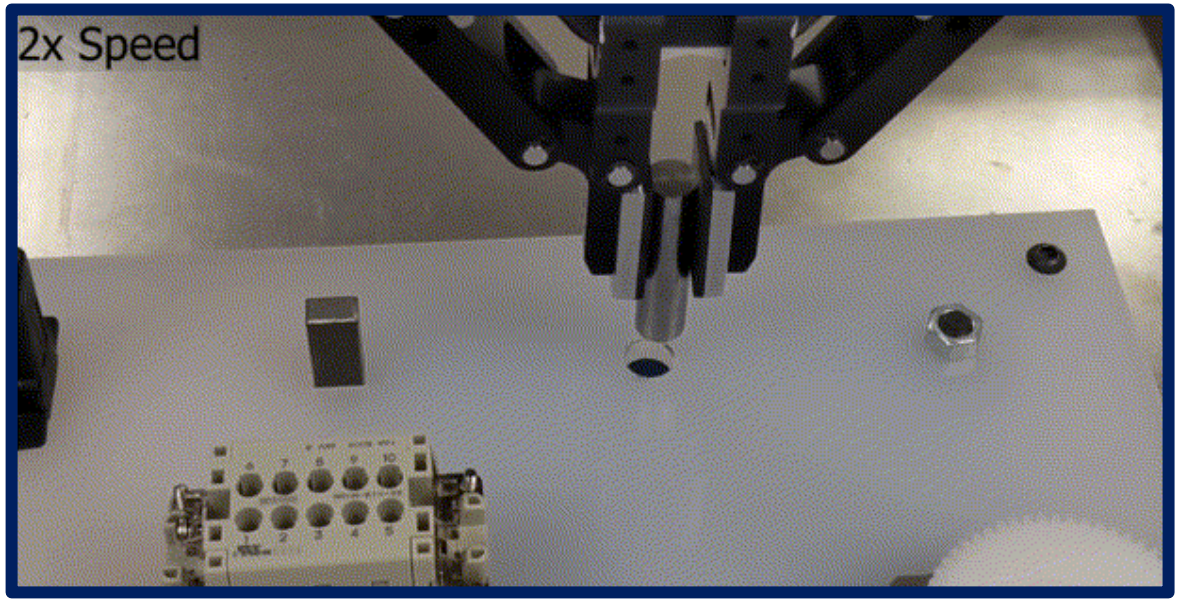
All Tasks needed for this application

Instantiate Interface and Conntext

Instantiate the Conntask, passing in Connfig, then run

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

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

    # initialize the Conntact environment by starting up the interface and conntext
    interface = ConntactROSInterface("conntact_params")
    conntext  = Conntext(interface)

    # Load the params for this example from the params file and process them in.
    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['connfig'])

    # ** 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(Fore.MAGENTA + "Node has control again!" +Style.RESET_ALL)
```

# Conclusion




Example program summary:

- 230 lines of code
- 2 YAML files

Upcoming goals:

- Full ROS 2 support
- More example applications