2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)
Beijing, China, November 6-9, 2018

Extended State Machines for Robust Robot Performance in Complex Tasks

Vinayak Jagtap vvjagtap@wpi.edu
Shlok Agarwal ssagarwal@wpi.edu
Sumanth Nirmal sgavarraju@wpi.edu
Sahil Kejriwal skejriwal@wpi.edu
Michael A. Gennert michaelg@wpi.edu

Abstract— **Most field robots today work under partial or complete guidance of an operator. The operator monitors, or at times augments, the control inputs of the robot to achieve better results or desired behavior. Robots that are operated remotely and over low bandwidth channels limit the involvement of the operator, leaving them vulnerable to unanticipated scenarios. The NASA Space Robotics Challenge (SRC), held in 2016-17, posed a challenge to operate a simulated Valkyrie R5 humanoid robot over a minimum bandwidth of 64-4k bits/second uplink, 50k-380k bits/second downlink, and a maximum latency of 20 seconds. To achieve this, we designed and implemented extended state machines that allow a robot to perform known tasks autonomously in a partially known environment along with the flexibility to perform system critical interventions manually, if required. The main intuition behind our approach is to combine (a) sensor data redundancy for object detection and (b) 2-stage motion planning approach using state machines to successfully accomplish complex tasks. The complex tasks demonstrated are aligning a communication dish, picking up a solar panel, and deploying solar panels autonomously. The overall system design allowed successful completion of tasks even after sub-task failures and/or complete communication loss.**

I. Introduction

The humanoid robotics community is striving to enhance development of ground robots that could perform complex tasks in dangerous, degraded, and human-engineered environments. A huge part of the motivation and momentum came from the DARPA Robotics Challenge (DRC)[1] that was held between years 2012-2015. Even though the competition sparked the interest of many roboticists, it indicated several shortcoming in the field of humanoid robotics. “Robots can’t do much, robots are slow, robots fall down” [2]. The NASA Space Robotics Challenge (SRC), which began in 2016, focused on developing software to increase the autonomy of humanoid robots so they can perform or assist with tasks during space travel or after landing on other planets (such as Mars), as well as on Earth. The Valkyrie R5 platform [3], built during the DRC, was used for this competition. Valkyrie is a 44 degree-of-freedom (DOF), series elastic actuator-based robot whose intended application is not only responding to events such as the Fukushima nuclear disaster, but also advancing human spaceflight and alien planetary endeavors in extraterrestrial settings. The tasks in the competition [4] were based on the conditions that might be found if a robot were sent to another planet to set up a base before humans arrive. One of the major challenges posed by space robots is slow or non-existent communication from the distant robot to earth. The distance between Mars and Earth adds high latency to radio signals which are already at low bandwidth. For example, with the Curiosity rover, which is one of the current active rovers on the surface of Mars, operators face one-way delays between 4 and 24 minutes. In NASA’s fifty years of Mars exploration, there have been numerous instances where communication has been lost with a probe or rover over extended periods of time [5]. For instance, in 2004 NASA lost complete communication with the Spirit rover on Mars which impaired its abilities. One way of tackling the communications delay problem is to move towards complete autonomy, which allows the system to operate during communication blackouts. In this paper, we as team WPI Humanoid Robotics Lab (WHRL), present our approach for introducing autonomy in humanoid robots for known tasks with the ability to control any predefined state through a low bandwidth and high latency connection. The sub-tasks in every task are in total order and can be retried if an attempt fails. We introduce two novel techniques to obtain reliability and robustness in our system. First we introduce new strategies for applying perception and manipulation in the extended state machine[6] to enhance the efficiency of the system. Secondly, we introduce a three mode operation to ensure reliability of the system by effective handling of the exceptions. The experimental results confirm that our methods ensure successful completion of tasks with little or no manual intervention. This approach is implemented in simulation on the Valkyrie R5 robot to perform complex tasks like turning handles to align a communication dish and picking up and deploying a solar panel.

In Section II, we provide a quick review of similar work. Section III describes the overall system that we designed and implemented to operate the Valkyrie R5 robot. Section IV is dedicated to the design of the state machine. The perception and manipulation aspects are covered in Section V and VI, respectively. We present our results in Section VII followed by conclusions in Section VIII.

II. Related Work

The work described here extends work in supervised autonomy for the DARPA Robotics Challenge [7] and [8], which used much tighter supervisor oversight than in this work. There, the default behavior when the robot sensed an impending problem was to stop and wait for operator intervention. Additional descriptions of levels of autonomy for drones can be found in [9] and for vehicles in [10]. A major distinction between these levels of autonomy and our

All authors are affiliated with the Robotics Engineering Program, Worcester Polytechnic Institute, 100 Institute Rd, Worcester, MA 01609, USA.

work is that they require a hard real-time response, whereas for a planetary robot, it may be acceptable to wait for human guidance. However, one does not necessarily want to wait too long, so trying alternatives before requesting operator intervention is likely to be a more reasonable course of action. [11] explains preliminary work with levels and layers to control a mobile robot and representation of behavior modules in a subsumption architecture. Application of state machine and decision making tools in robots can be found in [12], [13] and [14]. To tackle systems with a large number of states, a learning based approach to select state transitions is shown in [15].

III. System Overview

The entire system consists of 3 entities - Simulator, Field Computer, and Operator Control Unit (OCU) as shown in Figure 1. The field computer is connected to the simulator with a high-bandwidth low-latency connection, whereas the connection between field computer and OCU is over a low-bandwidth and high-latency network. The robot/simulator uses Robot Operating System (ROS) [16] and controllers developed by the Institute for Human & Machine Cognition (IHMC) [17] that provide an interface to control the robot using ROS messages. The software architecture of the field computer is designed to facilitate the implementation of the state machine which forms the basis of making the tasks autonomous. Figure 2 gives a brief overview of the structure of the APIs developed and installed on the field computer and how they interact with the simulator. An operator employs OCU to control the robot or to visualize sensor data. All experiments are conducted in the simulation using the NASA Valkyrie R5 robot. When the experiments are to run on real-time hardware, the simulator can be replaced by the real robot. In fact, the APIs have been tested on real hardware and have performed well in practice. In the rest of this section we explain the software architecture on field computer and the communication between field computer and OCU.

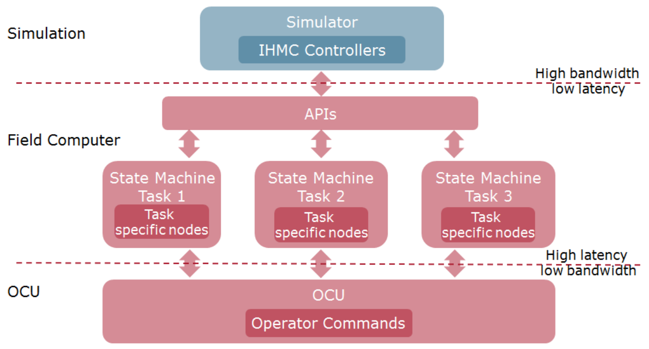

Fig. 1: System Setup

A. Perception APIs

Valkyrie R5 has a Multisense SL sensor that includes stereo cameras and a spinning LIDAR. The stereo cameras provide color images and distances with less accuracy and latency than LIDAR whereas LIDAR provides more accurate distance information with high latency and no color information. High latency in LIDAR data is due to the rotation of LIDAR which takes about 4 seconds for one complete revolution. The perception package contains libraries which are common to any kind of robot that uses Multisense SL sensor, which includes providing rgb/disparity images, raw pointcloud, and camera calibration parameters. *Laser Assembler* combines the received laser scans into a single pointcloud. This assembled pointcloud is used to maintain a master copy of pointcloud of the visible world. *Walkway Segmentation* finds planar regions and provides ground plane information. This is fed to the Navigation APIs for generating 2D occupancy grid.

We implemented detectors for detecting known objects required for completing specific tasks. These detectors use images and assembled pointcloud for detecting objects. Images provide colors and patterns on surfaces along with other features like edges, corners, etc. which can also be found in pointcloud. We use the information from both these sources to make the detection robust by allowing the system to retry detection if one succeeds and other fails.

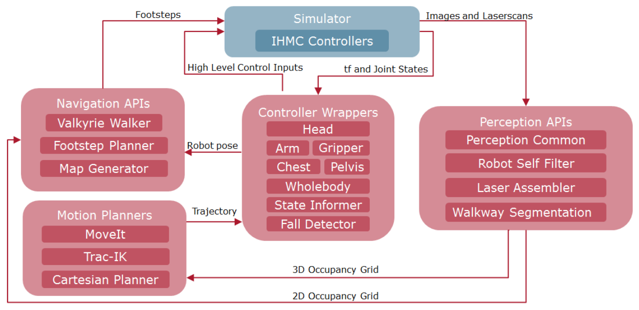

Fig. 2: Software Architecture

B. Navigation APIs

Map Generator creates a 2D occupancy grid using the segmented walkway from Perception APIs. This helps create a 2D map of planar regions where the robot can navigate. The *Footstep Planner* plans footsteps from the robots initial position to a goal position in the map using the ARA* planning algorithm [18]. These planned footsteps are converted into IHMC messages by *Valkyrie Walker* and sent to the robot.

C. Motion Planners

The motion planners are responsible for planning trajectories required for manipulation. To plan a collision free path, the planners receive a 3D occupancy grid in the form of an octomap which provides information of free and occupied areas in the environment [19]. Any type of search-based or sampling-based planner could be used in this framework to achieve desired trajectory points. The motion planners also contain an inverse kinematics solver which provides joint trajectories for every waypoint in the trajectory. These joint trajectories are then sent to the controller wrappers.

D. Controller Wrappers

This module provides wrappers over the IHMC controllers which interact with the robot. The wrappers can send commands to move individual joints of the robot or whole-body motions to perform complex tasks. The controller wrappers listen to the joint state and frame information of the joints using a ROS package, tf[20] to read the current state of the robot. They also receive trajectories from the motion planners and convert them into high-level control inputs which are sent to IHMC controllers to perform desired motion.

E. Communication Between Field Computer and OCU

Due to low-bandwidth network connection between the field computer and OCU, data transfer between the two should be limited. Even with bandwidth restriction, the operator should have access to the robot state with the ability to intervene and assist the completion of the task to ensure system safety. These capabilities are provided using one channel for sending/receiving text and one for graphical user interface (GUI) on OCU. Figure 3 shows both the channels where a chat server type configuration is used for text commands and a VNC [21] server and client is used for visualization. The chat server comprises of a broadcast server, a message parser, a reader and a writer. The broadcast server is an asynchronous chat server that transmits all the messages it receives to all the clients that are currently connected. The message parser converts messages sent by the writer into commands and executes these commands on the field computer, it also publishes log messages from the field computer. The reader client displays state machine log and robot state information on OCU. The entire visualization of sensor data from robot runs on the field computer using rviz[1], VNC server, and a windowing system. The operator can see the robot and sensor information using a VNC client on OCU. As VNC client can be tuned to communicate at low resolution images, this visualization approach reduces the data transmitted drastically by eliminating the need to transmit huge data structures like pointcloud and occupancy grids over the network.

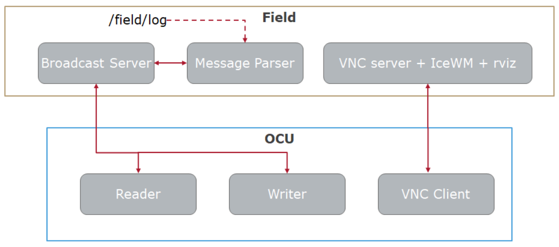

Fig. 3: Communication between field computer and operator control unit

IV. State Machines

A state machine is a mathematical model of computation. It consists of a set of states defined for a system which may or may not be independent of each other. At any given time, the state machine is in exactly one state and the transitions from one state to another are governed by either internal or external triggers. In this paper, we use the term 'state machine' interchangeably to refer to the mathematical model or to the program that implements it. We designed independent extended state machines for each task that is started by the operator. Our representation of the extended state machine is defined as an 8-tuple $M = (\Sigma, S, s_0, \delta, D, F, U, A)$ where,

- Σ is the set of input symbols, comprising all the inputs that are accepted by the state machine.
- S is a set of states.
- $s_0 \in S$ is the initial state for a task.
- D is a set of state variables that are used in the trigger function. Let b_i be a variable that stores the successful completion status of state i, ${c_i}^k$ be a set of variables that stores the count of failures while executing the path from state k to state i, n is the number of states and $h_i \in H$ is a set of hardware statuses, thus

$$D = \{\{b_i, {c_i}^k, h_i\} \quad | \quad 0 < k \leq i < n\}$$

- F is a set of trigger functions responsible to generate triggers for state transitions. It is defined as,

$$F = \{f_i : D \to \{0,1\} \quad | \quad 0 < i < n\}$$

  The individual trigger functions returns 1 only when the current state is successful, the failure count of each path is less than a specified threshold, and the robot is standing with no hardware errors.
- U is a set of update functions responsible for updating state variables.

$$U = \{u_i : D \to D \quad | \quad 0 < i < n\}$$

- A is a set of Actions.
- δ is the transition function responsible for transition from the current state to the next state when the trigger function evaluates to true. If the trigger function evaluates to false, the machine transitions to a previous state depending on the value of ${c_i}^k$. The state variables are updated using function U at the transition.

$$\delta : S \times F \times \Sigma \to S \times U \times A$$

We implemented this state machine for 2 tasks. The first task requires the robot to fix yaw and pitch of a communication dish. In this state machine, there are 18 states. The second task involves picking up of a solar panel, deploying it on a solar array, and plugging in power cable into the solar panel. This has a total of 30 states. Not all states are visited during a run. For each state machine, s_0 corresponds to the robot detached from its supporting harness and standing unassisted. b stores the status of successful completion of action $a \in A$, such as detection of object, reaching a specified goal, or attaining a required robot configuration. Once this is achieved transition function, δ, is triggered with the trigger function, f, of that state.

To ensure reliability we provide three different modes of operation and a non-recoverable failure mode as shown

[1] http://wiki.ros.org/rviz

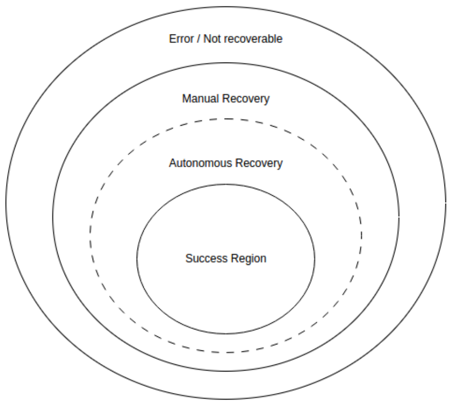

Fig. 4: Operational Modes of the State Machine

in Figure 4 which represents our approach to exception handling. The center of the figure is the region of success. This region is necessary for transition to the next state. In normal execution, the robot state will reside in this region. The second layer represents the space of all the trigger functions (F) and hardware statuses (H) in which the state machine is able to converge autonomously to the success region. The third layer is the space corresponding to manual execution mode in which the trigger functions fail and the state variables (D) are beyond the programmed logic and needs operator intervention to proceed further. The system moves into this layer when there is a non-critical issue and needs user attention to proceed. All the states in the second and third layer can converge to the success region with the transition function defined above. The system moves to the outermost layer when the robot is not controllable due to hardware or software limitations. For example, Valkyrie R5 cannot stand up if it falls and there is nothing an operator can do once the robot is flat on the ground. This situation ends up in the 'Not Recoverable' layer.

The design of transition functions is crucial for reliable handling of failures autonomously. Perception and manipulation being very different but interdependent operations, it is important to tune the state transitions for better results. The next section provides details of detection strategies using redundant sensor data to provide accurate object locations. Manipulation algorithms can use these object locations to perform required manipulation as explained in Section VI.

V. Perception

Object detection using images of known objects in an unknown environment can be performed using either computer vision techniques or learning algorithms. In both cases, known features of the object of interest are searched in the field of view to determine if the object is present. At times, it is possible that similar features are detected on different objects or the trained features of the desired object are not visible in the current perspective. Such instances lead to wrong detection or complete failure of detection. Other factors like lighting conditions and occlusions impact the accuracy of detection and add to the uncertainty. On the other hand, using a pointcloud for object detection is an expensive process due to processing a potentially large number of points. Even though pointcloud data obtained from LIDAR is more accurate, it inherently does not provide color information, which makes object detection difficult.

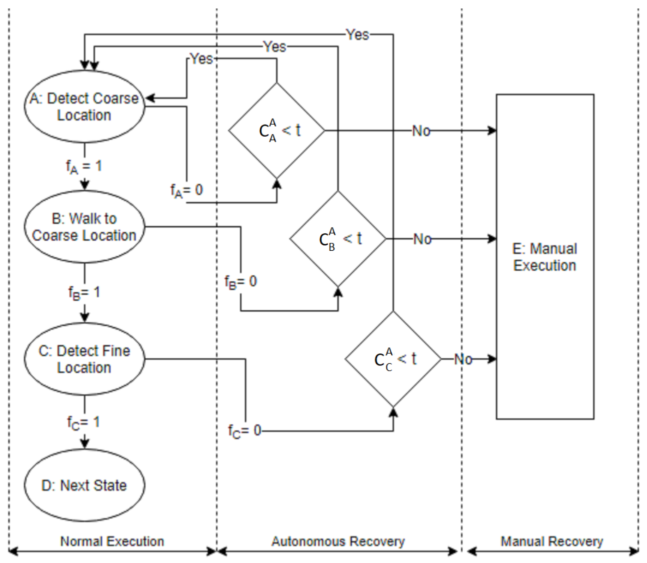

Fig. 5: Perception with Recovery Modes.

For better use of resources, it is important to leverage the accuracy of pointcloud data and the color of images to make detection robust and computationally inexpensive. We introduce an approach that takes advantage of both information sources by employing the state machine to perform coarse detection based on color images and then fine detection using pointcloud data. As shown in Figure 5, coarse detection is performed first. In this state, images are used to filter a region of interest (ROI). If the detection fails more than a set threshold, t, the state machine enters manual mode and the operator can either provide the location of the object as a marker or change the robot's configuration such that the probability of detecting the object becomes higher. After identifying the coarse goal, the robot navigates to a location determined by coarse detection and performs fine detection. Fine detection is performed by extracting shape features of the object from the pointcloud data in the ROI. We use a pass-through filter to extract the relevant volume of the pointcloud for further processing. This helps in faster feature extraction at lower computational cost. If coarse detection is wrong, fine detection would fail as the robot would search for that object in an incorrect location. When the fine detection fails less than the set threshold number of times, the system transitions back to coarse detection. This provides another trial for detection based on images to confirm or change the previous coarse goal. Based on the output of coarse detection, the robot might move from its position allowing a different perspective for fine detection.

VI. Manipulation

The success of manipulation tasks depends on both precise detection and navigation to ensure that the object of interest is reachable. Objects to be manipulated were known ahead of time but absolute position, orientation and surface friction coefficient of those objects can vary based on environment. Since we have partial information of the environment, we can avoid checking for collisions to speed up the planning and execution process of the task. After determining the position of the object, the robot would align itself with the object at a fixed predefined offset to move or pick the object. We generated a set of predefined standing and grasping poses for the robot for every manipulation task. Using the state machine framework explained in Section IV, different configurations are tried in case of failures.

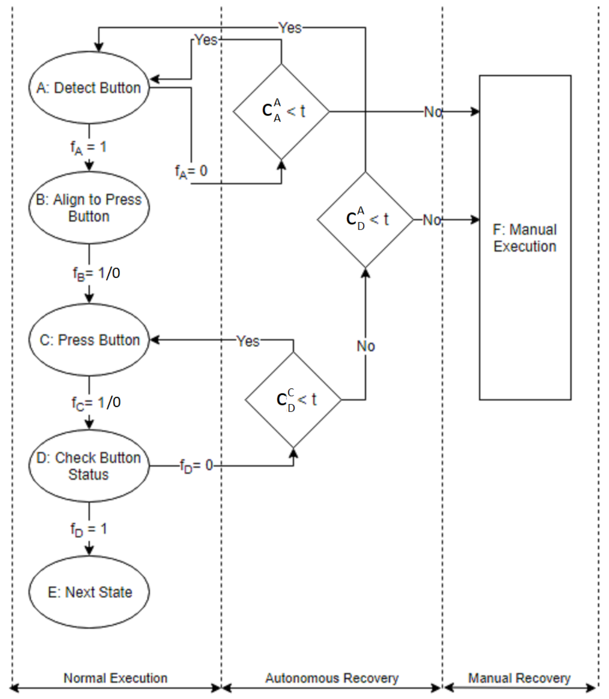

Fig. 6: Button Pressing Sub-task with Recovery Modes

To demonstrate the process of carrying this out using the approach explained in Section IV, a simplified example task of pressing a button is shown in Figure 6. The expanded representation of the *Detect Button* state is shown in Figure 5. Successful detection of the button leads to aligning the robot within reachable distance of the button. After aligning with the button, the robot proceeds to the *Press Button* state followed by checking the expected output after pressing the button. If the manipulation and perception APIs return results within a certain margin of error, the state machine operates in the normal execution mode. In cases where the robot receives an incorrect detection or is unsuccessful in pressing the button, the state machine falls into the autonomous recovery mode where using successive trials the robot tries to complete the task. In situations where the autonomous recovery mode fails to complete the task, the state machine transitions to a manual execution mode, which is the final layer of the state machine. In this mode, the operator must manually intervene to change the robot pose such that subsequent trials may succeed. Using these three modes, the robot can try multiple detections, standing poses, and motion plans before reaching out to the operator for assistance.

Humanoid robots have a floating base compared to the fixed base in other robotic manipulators. Thus, given a motion plan for a humanoid robot, we need to decide whether contact between the feet and the ground can be maintained. The IHMC momentum control framework [22] uses an underlying quadratic program which minimizes joint accelerations based on motion tasks, momentum objectives and ground reaction constraints. While generating motions, the quadratic program ensures that the Zero Moment Point (ZMP) or Centroidal Moment Pivot (CMP) [23] remains inside the support polygon made by the feet to maintain balance. This forces the robot to modify some trajectories to maintain balance irrespective of the task objectives. Such behavior ends up in failure of manipulation sub-task in certain standing poses but we can overcome the failure by retrying the plan with a different pose.

			% Mass			
Robot	**Mass (Kg)**	**Height (m)**	**Torso**	**Legs**	**Hands**	**Head**
Atlas	175.4	1.90	55.26	20.64	21.79	2.29
Valkyrie	135.2	1.88	35.69	36.59	21.92	5.73
iCub	32.5	1.0	42.53	37.37	14.52	5.56
Nao	5.3	0.57	19.78	45.51	21.81	12.88

TABLE I: Mass distribution of different sized humanoid robots

Table I shows the mass distribution for four humanoid robots of different sizes. We observe that a significant amount of mass resides in the torso for mid- and large-sized robots such as iCub, Atlas, and Valkyrie. This is due to their overall bulk and heavy battery packs. Assuming that a robot is in its default stance with the torso aligned vertically above the hip, it can be inferred that movement of the torso has the highest impact for change in Center of Mass (COM) and ZMP during manipulation in double support phase. Therefore the planner should try to minimize movement of the torso for better static stability.

While calculating the inverse kinematics (IK) solution for a task space point, the solution is biased with different joint weights. Nonetheless, we employ a simple generic two step planning process to make sure the solution is biased based on mass distribution of the robot as shown in Table I. In the first stage, we plan joint trajectories for the 7-DOF arm which moves the arm to the closest Euclidean distance from the object. In the second stage, we plan the remainder of the trajectory using the complete 10-DOF chain to achieve the desired position and orientation. This method provides a simple way to bias the IK solution based on mass distribution for mid-sized humanoids at the expense of increased planning time. We confirm the task completion based on visual inspection and/or task based ROS status messages.

	Percentage Completion				
State	**NE**	**AR**	**MR**	**#AR**	**Time**
Task 1					
Detect Panel - Coarse	100%	-	-	0	8.7
Walk to Panel - Coarse	100%	-	-	0	19.9
Detect Knobs - Coarse	100%	-	-	0	4.5
Detect Panel - Fine	90%	10%	-	1	10.6
Walk to Panel - Fine	100%	-	-	0	6.1
Detect Knobs	90%	10%	-	1	5.9
Grasp Pitch Knob	100%	-	-	0	13.3
Fix Pitch	40%	60%	-	9	36.7
Grasp Yaw Knob	100%	-	-	0	18.9
Fix Yaw	0%	100%	-	18	72.0
Detect Finish Box	90%	10%	-	1	46.9
Walk to Finish Box	100%	-	-	0	41.9
Task 2					
Detect Rover - Coarse	100%	-	-	0	8.6
Walk to Rover - Coarse	100%	-	-	0	25.2
Detect Rover - Fine	100%	-	-	0	0.6
Walk to Rover - Fine	100%	-	-	0	3.4
Detect Panel	60%	20%	20%	3	62.7
Align with panel	100%	-	-	0	1.6
Grasp Panel	100%	-	-	0	48.5
Pick Panel	100%	-	-	0	4.9
Align with Walkway	100%	-	-	0	24.8
Detect Solar Array-Coarse	80%	20%	-	1	8.5
Walk to Array - Coarse	100%	-	-	0	29.2
Rotate Panel	100%	-	-	0	6.5
Detect Solar Array - Fine	100%	-	-	0	8.9
Walk to Solar Array - Fine	100%	-	-	0	13.2
Place Panel	100%	-	-	0	20.6
Detect Button	100%	-	-	0	5.7
Deploy Panel	100%	-	-	0	26.3
Detect Cable	80%	20%	-	2	5.4
Pick Cable	100%	-	-	0	15.6
Detect Socket	60%	40%	-	3	5.8
Plug Cable	60%	20%	20%	12	105.8
Align with Walkway	100%	-	-	0	17.9
Detect Finishbox	100%	-	-	0	45.82
Walk to Finishbox	100%	-	-	0	24.19

TABLE II: Results for Task1 and Task 2.
Legend:
NE - Normal Execution **AR** - Autonomous Recovery
MR - Manual Recovery **#AR** - Total # of Auto Retries
Time - Avg. Time in simulation seconds

VII. Experiments & Results

Experiments are performed in a simulated environment with restrictions and tasks similar to what a robot might perform on a space mission such as on Mars. The operator cannot see the simulation environment and has to rely on messages that are transmitted to the OCU from the field computer. The first two tasks from the competition are used for the experiments. In the first task, the robot must detect a panel which controls the orientation of a communication dish. The robot must walk to the panel, rotate two handles to set the desired yaw and pitch alignment of the communication dish and then walk to the finish box. In the second task, the robot must detect and grab a solar panel. Then it must transport it to a solar array, deploy the panel, pick up the power cable and plug it into the deployed panel. Lastly, the robot must detect the finish position and walk to it.

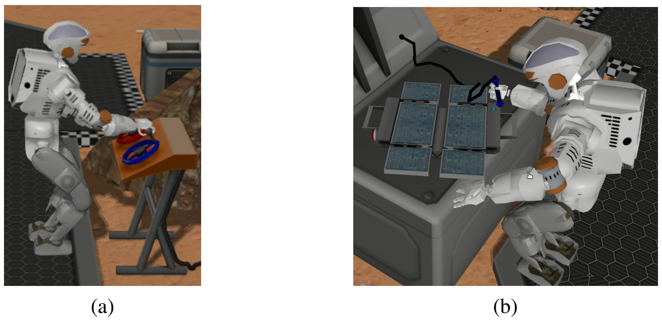

Fig. 7: Valkyrie R5 rotating communication dish knob in the left figure and plugging cable into solar panel in the right.

Before each run of a task during experiments, the environment is altered to have different object configurations, different model geometries, and varying friction coefficients, which makes each run unique. As shown in Figure 1, simulation and IHMC controllers are hosted on one machine and the developed libraries and state machines on another. The field computer and simulator are connected with a gigabit Ethernet connection. Network traffic between the field computer and OCU is restricted to a maximum of 380 bits per second with a round trip latency of 20 seconds. This environment setup and communication conditions were designed by competition organizers to emulate an environment (robot and field computer) on Mars with the operator (OCU) on Earth.

Table II aggregates the results of 10 runs for task 1 and 5 runs for task 2. The first column of the table is the current state of the state machine. As mentioned in Section IV, if the state is executed successfully, it should complete through one of the three modes: normal execution mode, recovery mode or manual execution mode. These are represented by the three columns under percentage completion for each task. For instance, row 4 can be read as state *Detect Panel - Fine* completed 90% of the time in Normal Execution mode and 10% of the time in Autonomous Recovery mode. 'Total # of Auto Retries' column lists how many times the state failed in the Autonomous recovery mode and had to be retried again. 'Average Time (s)' is the simulation time required per run for successfully executing a state. Task 1 was completely autonomous and did not require any manual intervention from the operator. It can be inferred from the 'Average Time' column that this approach is time-efficient as providing manual input with a round-trip latency of 20 seconds would have taken each state more than 20 seconds to complete. In cases when the state has to be retried, like the one in state Fix Yaw due to losing grasp of the handle, it would have taken 20×1.8 seconds additional per iteration for communicating between the OCU and field computer. The effect of manual execution can be seen in results of state *Plug Cable* which normally takes less than 60 seconds if completed in autonomous recovery mode, but due to one run out of five which required manual execution, the average time for 5 runs went up to 105.88 seconds.

The involvement of the operator in manual recovery mode is primary to the success of tasks which could not be handled by the normal or autonomous modes of operation. The operator has the ability to send manual commands defined in the APIs to finish the task in absence of the state machine, however, the time required would be dependent on the latency of communication and expertise of the operator controlling the robot. Operator error being one of biggest performance limitations[7], providing commands to perform a set of operations like move to a location, re-detect an object, switch to a specific state, etc. lets the robot make decisions of planning and validating motion trajectories instead of the operator.

Two major shortcomings of the simulation environment provided were that the robot could not stand up after falling down and it could not pick something up from the ground, which restricted the flexibility and operational space. Situations when the robot falls down or drops a tool on the ground result in a complete failure of the task. Such tasks could only be automated with system improvements like adding capabilities to the robot. This would require both software and hardware upgrades.

VIII. Conclusion

In this paper, we formalize an extended state machine to perform complex tasks for humanoid robots. We successfully demonstrated how we boost the reliability of the system by introducing a novel mechanism of three mode operations to manage state transitions. The state machine does not eliminate the need to write good perception or manipulation algorithms, but does boost the performance of existing algorithms by suppressing incorrect outputs and retrying under similar or different conditions. Our experiments confirm that our method can be successfully used to execute any predefined task with the proposed structure. Since all experiments were conducted in unique environments, successful completion of the tasks exhibit the robustness and reliability of the designed system and takes a step towards increased autonomy of humanoid robots.

ACKNOWLEDGMENT

We would like to thank NASA, NineSigma, OSRF and Space Center Houston for organizing the SRC and giving us an opportunity to work on this project. We would like to acknowledge the Dean of Arts and Sciences, Dean of Engineering, CS, ME, ECE departments and the Robotics Engineering program for their resources and constant support during the competition.

References

[1] G. Pratt and J. Manzo, "The darpa robotics challenge [competitions]," *IEEE Robotics & Automation Magazine*, vol. 20, no. 2, pp. 10–12, 2013.

[2] C. G. Atkeson, B. Babu, N. Banerjee, D. Berenson, C. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, *et al.*, "What happened at the darpa robotics challenge, and why," *submitted to the DRC Finals Special Issue of the Journal of Field Robotics*, 2016.

[3] N. A. Radford, P. Strawser, K. Hambuchen, J. S. Mehling, W. K. Verdeyen, A. S. Donnan, J. Holley, J. Sanchez, V. Nguyen, L. Bridgwater, *et al.*, "Valkyrie: Nasa's first bipedal humanoid robot," *Journal of Field Robotics*, vol. 32, no. 3, pp. 397–419, 2015.

[4] (2016) Space robotics challenge - rules. [Online]. Available: https://perma.cc/RN55-9R6Y

[5] Nasa mars exploration timeline. [Online]. Available: https://nssdc.gsfc.nasa.gov/planetary/chronology%5Fmars.html

[6] K.-T. Cheng and A. S. Krishnakumar, "Automatic functional test generation using the extended finite state machine model," in *Design Automation, 1993. 30th Conference on*. IEEE, 1993, pp. 86–91.

[7] C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, M. Gennert, J. P. Graff, P. He, A. Jaeger, J. Kim, K. Knoedler, L. Li, C. Liu, X. Long, T. Padir, F. Polido, G. G. Tighe, and X. Xinjilefu, "No falls, no resets: Reliable humanoid behavior in the darpa robotics challenge," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov 2015, pp. 623–630.

[8] N. Banerjee, X. Long, R. Du, F. Polido, S. Feng, C. G. Atkeson, M. Gennert, and T. Padir, "Human-supervised control of the atlas humanoid robot for traversing doors," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE, 2015, pp. 722–729.

[9] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, 05 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14542

[10] N. H. T. S. Administration *et al.*, "Preliminary statement of policy concerning automated vehicles," *Washington, DC*, pp. 1–14, 2013.

[11] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[12] C. Armbrust, D. Schmidt, and K. Berns, "Generating behaviour networks from finite-state machines," in *ROBOTIK 2012; 7th German Conference on Robotics*, May 2012, pp. 1–6.

[13] P. Allgeuer and S. Behnke, "Hierarchical and state-based architectures for robot behavior planning and control," in *Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conf. on Humanoid Robots, Atlanta, USA*, 2013, pp. 3–5.

[14] M. Foukarakis, A. Leonidis, M. Antona, and C. Stephanidis, "Combining finite state machine and decision-making tools for adaptable robot behavior," in *International Conference on Universal Access in Human-Computer Interaction*. Springer, 2014, pp. 625–635.

[15] B. Argall, B. Browning, and M. Veloso, "Learning to select state machines using expert advice on an autonomous robot," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2124–2129.

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[17] SylvainBertrand, J. Pratt, D. Calvert, G. Wiedebach, J. Smith, T. Koolen, R. Griffin, N. Mertins, D. Stephen, neyssette, S. McCrory, mahopkins, jcarff, K. Cesare, egalbally, mj fl, P. Abeles, edilee, pneuhaus, A. B. Filho, W. Rifenburgh, N. Choe, K. Petrnek, OlgerSiebinga, lbunch, K. Krmer, V. Ivan, admimou, aprvs, and turbancov2, "ihmcrobotics/ihmc-open-robotics-software: SRC Finals," July 2017. [Online]. Available: https://doi.org/10.5281/zenodo.827342

[18] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2012.

[19] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9321-0

[20] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

[21] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998.

[22] T. Koolen, S. Bertrand, G. Thomas, T. De Boer, T. Wu, J. Smith, J. Englsberger, and J. Pratt, "Design of a momentum-based control framework and application to the humanoid robot atlas," *International Journal of Humanoid Robotics*, vol. 13, no. 01, p. 1650007, 2016.

[23] M. B. Popovic, A. Goswami, and H. Herr, "Ground reference points in legged locomotion: Definitions, biological trajectories and control implications," *The International Journal of Robotics Research*, vol. 24, no. 12, pp. 1013–1032, 2005.