In Proceedings of the ACM/IEEE Fourth International Conference on Cyber-Physical Systems (ICCPS-2013) 2013.

Architecture of a Cyberphysical Avatar

Song Han*, Aloysius K. Mok*, Jianyong Meng*, Yi-Hung Wei*, Pei-Chi Huang*, Quan Leng*
Xiuming Zhu*, Luis Sentis†, Kwan Suk Kim†, Risto Miikkulainen*

*Department of Computer Science, The University of Texas at Austin
{*shan, mok, jmeng, yhwei, peggy, qleng, xmzhu, risto*}*@cs.utexas.edu*
†Department of Mechanical Engineering, The University of Texas at Austin
{*lsentis@austin.utexas.edu, kskim@utexas.edu*}

***Abstract*— This paper introduces the concept of a cyberphysical avatar which is defined to be a semi-autonomous robotic system that adjusts to an unstructured environment and performs physical tasks subject to critical timing constraints while under human supervision. Cyberphysical avatar integrates the recent advance in three technologies: body-compliant control in robotics, neuroevolution in machine learning and QoS guarantees in real-time communication. Body-compliant control is essential for operator safety since cyberphysical avatars perform cooperative tasks in close proximity to humans. Neuroevolution technique is essential for "programming" cyberphysical avatars inasmuch as they are to be used by non-experts for a large array of tasks, some unforeseen, in an unstructured environment. QoS-guaranteed real-time communication is essential to provide predictable, bounded-time response in human-avatar interaction. By integrating these technologies, we have built a prototype cyberphysical avatar testbed.**

I. Introduction

The utility of teleoperated robotic devices in mission-critical tasks is undisputed. Two examples in recent years are the oil pipeline spill in the Gulf of Mexico and the search and rescue operation after the level 9.0 earthquake in Japan. Yet, much more can be accomplished if the teleoperated devices have more autonomous capabilities. Indeed, the underwater repair operation in the Gulf of Mexico was suspended and precious time was lost in stopping the oil spill when two of the robotic vehicles collided with each other. In the case of the Japan earthquake, critical repair operation had to be suspended when radiation level became too high for human safety. Despite the impressive progress by the robotics community in recent years, we are still quite a way from being able to trust fully autonomous robots to carry out mission-critical and safety-critical operations by themselves. On the other hand, today's unintelligent teleoperated devices cannot be counted on to perform well in physically difficult and unstructured environments. Short of a gigantic leap in technology that creates intelligent fully autonomous robots capable of functioning in an unstructured environment, we propose to chart a pathway to evolve the capability of teleoperated robotic devices from primitive mechanical remote-control to trustable autonomy and more intelligent teleoperation. Rather than trying to build fully autonomous robots from scratch, we do it gradually through less and less human teleoperation, all the while deploying these robots in actual tasks. This is a new and different approach, and likely to result in practical applications and advances much sooner, and in more robust and better adapted autonomous robots in the end.

A short version of this paper appeared in International Workshop on Real-Time and Distributed Computing in Emerging Applications [1].

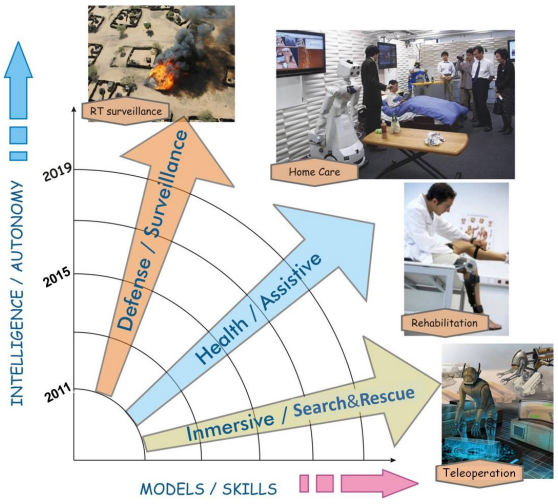


Fig. 1: **Vision of cyberphysical avatar technology:** The goal of cyberphysical avatar technology is to increase robot autonomy in unstructured environments. At the simplest level, robots are teleoperated with simple interaction and coordination capabilities, while by using a combination of models, skill learning, and communication-based switching between teleoperation and autonomous modes, they could lead to complex yet practical robot behaviors for applications in disaster, health, and security scenarios.

This paper describes such a pathway by introducing the concept of a "cyberphysical avatar". We define a cyberphysical avatar to be a semi-autonomous robotic system that adjusts to an unstructured environment and performs physical tasks subject to critical timing constraints while under human supervision. A cyberphysical avatar is semi-autonomous in that there are actions it must take without human intervention because of the relatively short timing constraints, e.g., the control loop that maintains a fast walking gait. On the other hand, a cyberphysical avatar should not be programmed to deal with only a fixed set of scenarios because we cannot foresee all the contingencies in all operational environments, e.g., a building on fire in a rescue mission. An effective interface between the cyberphysical avatar and its human supervisor is essential for success, and this requires the cyberphysical avatar to be designed for predictable and timely response. As the cyberphysical avatar gains more physical skills, it can be trusted to perform more subtasks on its own.

There are many technical challenges in realizing the cyberphysical avatar concept. These challenges can be categorized into three topics below:

- **Dynamics and control of humanoid avatars:** The cyberphysical avatar must be able to perform the physical tasks in an unstructured and uncertain environment. In this

area, we need to develop a methodology for modeling the dynamic behavior of physical avatars interacting with unstructured environments and controllers that can adapt to the changing physical conditions. We need to develop software foundations that encapsulate the physical skills supporting the full range of teleoperated to autonomous behaviors.

- **Evolutionary learning of skills under environment and performance constraints:** Evolutionary approach is needed to learn the continuous control parameters of the skills, as well as their discrete composition. The cyberphysical avatar must be able to acquire skills so that it can perform time-critical tasks autonomously. The level of autonomy and the criticality of the timing constraints that can be satisfied for practical physical tasks requires advances in learning theory and engineering validation. Moreover, learning strategies need to be applied to learn the tradeoff between teleoperation and autonomy.
- **Supporting reliable, real-time avatar-human communication:** The cyberphysical avatar must be able to operate untethered and maintain timely and reliably communication with the controller that is hierarchically implemented with the human supervisor at the top of the control hierarchy. The combination of real-time and robust communication in a wireless environment where communication paths may be disrupted requires advance in both algorithm design and engineering validation. We need a switching policy between teleoperated and autonomous behaviors that is based on communication quality as the primary metric for making switching decisions.

The rest of this paper will describe the system architecture of the cyberphysical avatar and some performance results of our implementation using the Dreamer/Meka humanoid robot. The cyberphysical avatar is fully operational at this point; but we envision much more new research and new ideas to be pursued in order to perfect the collaboration between the cyberphysical avatar and human supervisor.

II. Control of wheeled humanoid avatars in unstructured environments

The cyberphysical avatar must be able to maneuver in irregular terrains while performing accurate physical whole-body compliant interactions with the environment and with human operators. To attain these capabilities, skill modeling and control in unstructured environments must be carefully designed. In this section, we first describe the dynamic model of the wheeled base of our Dreamer/Meka humanoid robot under varying contact conditions. We then present our model of the whole-body compliant skill of the robot and the hierarchical control structures that is used to handle task conflicts during the execution of the behavior.

A. Dynamic model of the wheeled base

In unstructured and uncertain environments, wheel-based avatars will often be in a situation of marginal contact, i.e. not all the wheels are in contact. As such, the dynamics of the robot, need to represent the contact state of the robot, its effect on center of mass balance and the conservation of angular and linear momentum due to marginally-stable contact conditions. This characteristics become even more critical when the robot engages into manipulation tasks while maintaining marginal contacts.

We derive the model of the robot under varying contacts by leveraging the generalized contact consistent Jacobian developed in [2] which specifies that for a given contact state C_m the generalized Jacobian of an operational task (e.g. one of the robot's hands) is equal to

$$J^*_{\text{task},C_m} \triangleq J_{\text{task}}\ \overline{UN}_{C_m}, \tag{1}$$

where J_{task} is the Jacobian of the hand Cartesian point with respect to an inertial frame outside of the mobile base, U describes the underactuated (i.e. uncontrollable directions) of the base due to the contact state, N_{C_m} describes the current contact state (i.e. how many wheels are in contact), and the operator $\overline{(.)}$ indicates a dynamically consistent generalized inverse of the argument. Therefore the control of the operational task (e.g. the control of the robot's hand) while taking into account the mobility of the base and the uncertain contact state is equal to

$$\Gamma_{C_m} = J^{*T}_{\text{task},C_m}\ F_{\text{task}} \tag{2}$$

where F_{task} is the force or impedance command to control the hand, J^*_{task,C_m} is the whole-body task Jacobian including the base contact state (i.e. how many wheels are in stable contact), and Γ_{C_m} is the whole-body command of torques sent to the base and upper humanoid torso motors.

B. Skill definition and hierarchical control structure

In whole-body compliant control (WBC), a task is defined via a mapping between the robot's N-dimensional joint configuration and some M-dimensional space which describes an objective that the controller should achieve. The skill is defined as a juxtaposition of multiple operational tasks to help translate between high-level goals (such as provided by planning algorithms) and the operational tasks. In our environment, a skill is a human readable file (e.g. YAML) describing the points or coordinates of the robot that are to be simultaneously controlled to accomplish a behavior, plus their respective control policies, and plus their hierarchical priorities in the execution pipeline. In this work, however, as to be elaborated in Section III, the control policies of the skill will be learned through machine learning approaches.

Having now many operational task processes to simultaneously optimize, as defined in the skill, either as force or impedance processes, we propose to use the following control structure in Eq. 3. The intuition behind this control structure is to instantiate several tasks, each of which tries to drive the robot toward some state. The task contributions are accumulated using null space projections to ensure that lower-priority tasks do not interfere with higher levels. The motion is thus determined by each task in combination with their priorities. This structuring provides two orthogonal ways of changing robot behavior, either by influencing the tasks (e.g. changing their gains or goals) or by rearranging the hierarchy (e.g. inserting tasks or locally inverting their ordering).

$$\Gamma_{C_m} = \sum_k \left(J^{*T}_{k|\text{prec}(k),C_m} F_k \right) + N^{*T}_{t,C_m} \Gamma_{\text{posture}} + J^{*T}_{i|l,C_m} F_{\text{int}}, \tag{3}$$

In Eq. 3, F_k is the task space force or impedance command for the k-th operational task, $J^*_{k|\text{prec}(k),C_m}$ is the prioritized contact

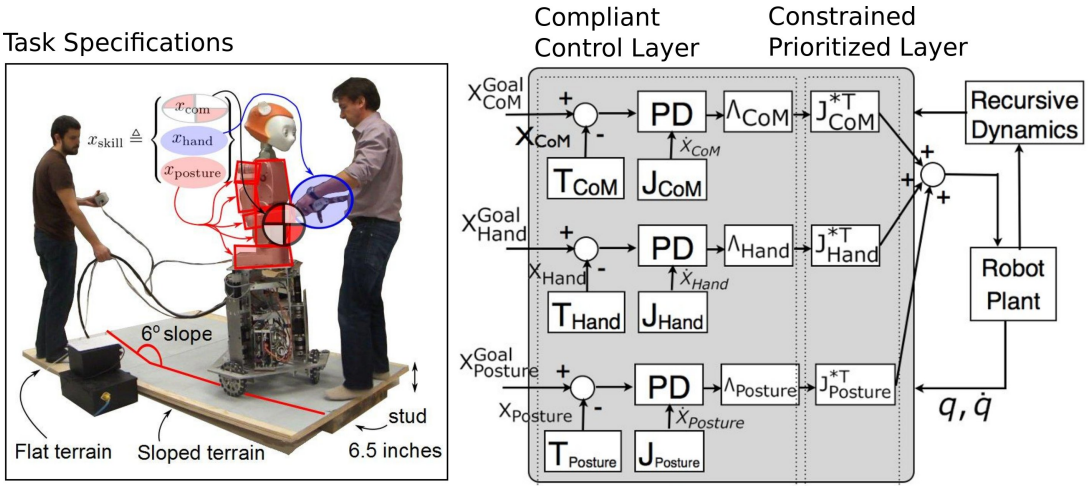


Fig. 2: Whole-body compliant control with prioritized tasks. Left hand side of figure shows the Dreamer crosses a terrain with a slope while responding to human interaction. And right hand side of the figure shows closed loop dynamic controller producing joint torque outputs based on Center of mass, hand position and posture with prioritized Jacobians.

consistent Jacobian of the task, Γ_{posture} is the command to optimize the posture behavior, F_{int} is the command to optimize the internal forces between the arms and the mobile base, and $J^*_{i|l,C_m}$ is the Jacobian of the internal forces. This structure is a derivation of our previous work on whole-body compliant control found in [3].

In the control structure, the low-level tasks describing the skill are aggregated using a hierarchy, where more relevant tasks, such as those who ensure the fulfillment of physical constraints appear first, while those dealing with the operational behavior appear with less priority. Fig. 2 gives an example where the skill is composed of three tasks. The first task is maintaining coordinates of center of mass(CoM) to prevent the robot from falling down on irregular terrain. Second task is compliant hand position which enables the robot to respond compliantly to human interaction. The posture task here is utilizing remaining degree of freedom to stabilize self-motion and converge to a human-like posture. Lower-priority tasks operate in the null space of all higher priority tasks. So when the terrain changes the CoM task will temporarily override non-critical tasks in order to prevent falling. The task becomes unfeasible when the current higher priority tasks use all the dynamic redundancy. This event can be easily monitored and used to stop the behavior and communicate the problem to a high level planner.

Because our control structures use effectively the dynamic and contact model of the physical avatar in its environment, they are able to optimize all task processes simultaneously within the contact stance, thus achieving precise tracking of forces and trajectories. Moreover, posture behavior, which is specified as an optimization criterion instead of a trajectory is also optimized within the residual manifolds left over by the priority tasks.

III. Skill acquisition by machine learning

Although much of the operation of the robot can be based on carefully designed control algorithms, there are two issues where machine learning methods can prove crucial: (1) conversion of human operator behaviors to robot behaviors, and (2) optimization of robot behaviors. In both of these cases, it is possible to come up with measures of how good the behaviors are, but the optimal behaviors are not known. Therefore, machine learning methods based on exploration need to be used. In this section, a particularly powerful such a method, neuroevolution, is described first, followed by its application to train the learning skills of the grasper on the Dreamer humanoid robot to pick up objects with any shape.

A. Learning robust nonlinear control through neuroevolution

In the neuroevolution approach, evolutionary optimization method such as a genetic algorithm is used to construct the structure and the connection weights of a neural network so that the network performs as well as possible in a given task [4]. The neural network can be recurrent, implementing a sequence memory, and thereby making it possible to use the approach to discover sequential behaviors such as robot navigation, arm control, and grasping.

Neuroevolution differs from other machine learning methods in two important ways. First, most such methods are supervised, which means that they learn behavior that approximates a given set of examples [5]. The examples need to be carefully constructed to represent correct, or optimal performance—the learning system will then learn a function that interpolates between them smoothly. For instance, in grasping, a number of grasping situations (configuration of the hand, shape and position of the object) need to be created together with the optimal grasping behavior. In general, it is difficult to come up with such examples because optimal behavior is often not known; also, it is difficult to cover all possible situations, making the behavior incomplete. In contrast, neuroevolution learning is based on exploration and reinforcement: a population of neural networks is evolved through crossover and mutation, directed only by how well each network performs. It is thus possible to discover successful behaviors that human designers would find difficult to construct, and behaviors that are more general.

Second, other methods that are designed to learn under sparse reinforcement, such as Q-learning, or value function learning in general, assume that the current state of the system is completely known [6]. If objects are occluded, or the situation is changing, it is difficult for them to anticipate what will happen (because the observed values of actions cannot be associated with the correct state). In contrast, the neural networks employed in neuroevolution can disambiguate the state based on their sequence memory. Previous sensor values are part of the state

representation, making it possible to understand how the world is changing, and how to respond to it optimally. Such an ability is particularly useful in domains such as grasping that are highly dynamic and where sensors are limited.

The neuroevolution approach can thus be used as a training mechanism for the Dreamer robot as well. In particular, it is well suited for learning skills such as picking up an object. In the following, we will first describe the physics of the grasper on the Dreamer/Meka humanoid robot and then present the training details. Training is done following the NEAT [7] approach. A detailed description of the NEAT approach can be found in our technical report [8].

B. Physics of the grasper

We simulate the Meka hand using GraspIt! [9], an open-source grasping simulation environment developed at Columbia University. GraspIt! provides mechanisms for simulating gravity, interactions between rigid objects, physical modeling, and joint movement for the specific purpose of developing efficient robotic grasps. Robotic components in particular are modeled as a series of degrees of freedom specifying parameters such as default rotational velocity, maximum torque, and relative position and rotation ranges. Each degree of freedom is connected to a kinematic chain and associated with a visual model in the three-dimensional environment. These individual components are combined to create an entire robotic apparatus.

The Meka hand in particular is defined by one degree of freedom for each knuckle in each finger, as well as degrees of freedom for the thumb's rotator. The mechanics of this model will be modified in our studies in order to account for two phenomena. First, we wish to control the wrist, which isn't modeled explicitly by default. We will therefore add a wrist component to the Meka model supplied by GraspIt!. Second, most of the degrees of freedom in the Meka hand are not actuated. Each finger consists of three joints, which are all connected by a single rubber tendon. When the finger curls, all three knuckles curl in unison. We will therefore adjust the torques that we feed to the simulator to account for this interdependent joint behavior. A set of torques given to a single finger will conform with one another such that they are all equivalent to torques initiated by a stretching of the rubber tendon, which we see in the real robot.

C. Training the grasper

To properly grasp with the Meka hand, first, we have to design an input and output layers of our target neural network, as well as a fitness function to allow a gradual climb toward an efficient grasp. Because our Meka unit is primarily controlled using the Whole-Body Control Framework, the network is only responsible to directly manipulate orientation and position of the wrist. Thus, we designate an output node for each of these degrees of freedom.

Each neural network generated from NEAT receives several input data that illustrate our current state of the robot in an environment. Designing the input layer is less trivial. It is necessary to encode the entire state of the grasp in some way, which includes positions of the grasped object, as well as the object's shape. To reduce dependency on the shape of the grasped object, we encode the object's state by simply taking a depth map from the robot's Kinect sensor and assigning each depth data a unique input node in the neural network. In this way, the network is able to associate state of an arbitrary object in an arbitrary environment with a grasping strategy namely appropriate position and angle of the robot hand.The output of each network through NEAT is to predict where an object is and what is the best direction to grasp the object in the form of three-dimension hand positions and orientation.

The final stage of our design is to construct an adequate fitness function. The fitness function of a network n with respect to a corresponding object o is computed as the reciprocal of mean square error $\overline{M}$, the summation of distance between the center of robot's palm and a desired object, and also combined with the grasp quality metrics provided by GraspIt!. Let P_i be the predicted position of hand for grasping by the network, where $i \in x, y, z$ coordination. Let O_i be a coordinate of the selected object after mouse click, where i is captured from the Kinect sensor inputs to get x, y, z coordination. Let q be a quality value after the execution of a single grasp, which is normalized into the range $[0, 1]$ by GraspIt!. Thus, the fitness function f is defined as follows:

$$f(n) = \frac{\beta}{\overline{M} + \alpha} + \gamma q = \frac{\beta}{\sum_{i \in x,y,z}(P_i - O_i)^2 + \alpha} + \gamma q \qquad (4)$$

, where α, β and γ are constants.

The first term of the equation is measuring the distance between the hand and the object and the second term is proportional to the grasp quality. During the initial phases of learning, the coordinate is arbitrarily chosen, so the robot hand will try to grasp at arbitrary position where it can not even touch the object. As a result, the second term of the fitness function will be almost always zero in the early generations. So in this stage, the first term is used to differentiate the fitness of the neural networks, which will train the networks to get closer to the target object. After some generations, when the hand can grasp the object the second term will start be effective and rank the results according to the grasp quality. And parameters α, β and γ is used to adjust the relative affect of these two terms. So with this fitness function, the neural networks will first learn to get close to the object and then learn to grasp the object in right way.

D. Simulation results

To explore the robot behavior in the real world, we adopt dynamic environment in the simulation. Because GraspIt! only has limited support for the hand model of Meka robot and only provides restricted hand movement, we have improved the hand model and movement functionality to make it work with our simulation environment. Fig. 3 illustrates the overview of our simulation setup. The Kinect sensor is simulated within GraspIt! simulator which is used to provide a set of depth data as inputs for the neural networks. This array of depth data together with a two-dimensional coordinate representing mouse click from users are fed to the input layer of the NEAT. The output of the NEAT consists of hand position and hand orientation where they are sent back to the simulator for manipulating the robotic hand and evaluating the quality of grasp. The structure and the weights of the neural networks are automatically created with NEAT [7] through several generations.

The evolved neural networks are utilized to retain hand position and orientation, and we use these data to manipulate Meka hand in simulation environment for performing grasping and evaluating the quality of grasping. In this experiment, the

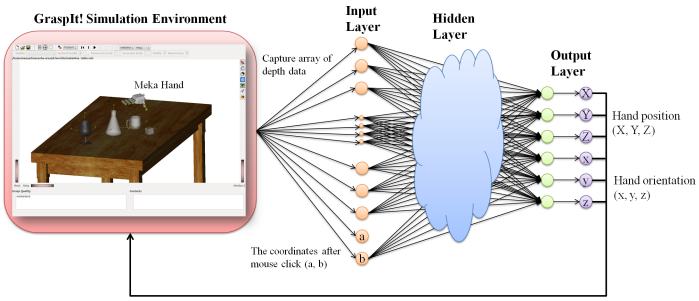


Fig. 3: Representation of the designed grasp controller network.

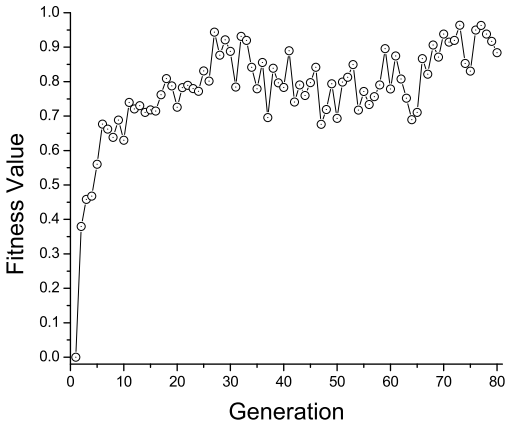


Fig. 4: The average maximum fitness function of the neural network found at each generation in a GraspIt! environment.

input data contains 20×20 depth array nodes, 2 coordinate nodes. The coordinate node here denotes mouse click input from the user indicating the target object. So the remote human controller can click on Kinect Sensor video stream to identify the target object of the grasping. In our experiment, this coordinate is created by randomly picking a point of the target object. These input data are directly connected to the output to form the first generation of neural networks, which are then evolved through mutation and crossover. We set the population size as 128, refine three parameters α, β and γ of the fitness function Eq. 4, and choose number of generations as 80. The fitness value is a function of the distance between the Meka hand and a target object plus the quality of grasping. In the fitness function, larger fitness value implies better grasping quality. As Fig. 4 shown, in the starting networks, the first average fitness is low because they are composed of the input data directly connected to the outputs with random weights. As the neural networks evolve through generations, performing adaptive weight and structure adjustment, the fitness value increases. After around 30 generations, it reaches around 0.8 which means the Meka hand can grasp the object more accurately with the proper position and orientation. We also observe that after 30 generations, the fitness value oscillates around maximum value. It is possible that we do not perfectly characterize our fitness function, and more fine tune shall be made to further improve the result.

E. Transitioning from simulated to physical controller

The training method described in Section III-C is primarily done in simulation. However, transferring controllers evolved in simulation to the physical robot is challenging [10], [11]. The main reason is that it is difficult to simulate physical properties such as friction and sensor and actuator characteristics with high enough fidelity to reproduce the simulated behaviors on real robots. In order to address this issue, we choose the following methods to improve the results of transfer to the real robot.

First, if the simulator is accurate enough, controllers that transfer well can be created simply by evolving them to be robust. That is, if sensor values and actuator responses frequently vary in simulation, the resulting controllers will be robust against small discrepancies between simulation and reality as well [12], [13]. Such uncertainties can be introduced into the simulation simply as noise, and solutions evolve that do not depend on accurate values and outcomes, thus transferring well.

If there are more systematic flaws in the simulation, behaviors may evolve that exploit them, and therefore transfer poorly. Such behaviors can be discouraged by incorporating transfer into evolution explicitly, by utilizing a multi-objective evolutionary algorithm that optimizes both a task-dependent controller fitness as well as a measure of how well the controller transfers from simulation to reality [14]. In any given generation, this method chooses at most one controller based on behavioral diversity to be evaluated on the real robot, requiring only a small number of hardware evaluations.

Another approach is to perform experiments on the real robot in order to improve the simulator, typically in one of two ways: (1) Experiments are performed on the real robot before running evolution to collect samples of the real world by recording sensor activations [15]. When controllers are evaluated later during evolution, these samples are utilized to set the simulated sensor activations accurately. (2) Experiments are performed on the real robot during evolution to co-evolve the simulator and the controller, making an initially crude simulation more and more accurate [16].

We will adopt the system-level simplex architecture [17] to provide safety guarantees during the transitioning. In this architecture, we use a simple and verified safety controller to ensure the stability and safety of the robot operations. This conservative safety control core is then complemented by a high-performance complex controller, which will be used whenever possible, but switch to the safety controller when system integrity is jeopardized.

IV. Supporting Remote, Reliable and Real-Time (R^3) Avatar-Human Communication

A key component of cyberphysical avatar technology is the remote, reliable and real-time avatar-human communication. Because of the mobility requirement of avatars to work in unstructured environments as is often the case in disaster recovery, wireless connection has to be established at the edge of the communication infrastructure. Owing to limited wireless communication range (e.g. around 200m in Wi-Fi and 40m in 802.15.4-based [18] low-power wireless standards), a multi-hop wireless network in mesh topology is required to cover a large area, and more importantly, can overcome transmission blocking by objects such as metal doors in industrial facilities where avatars operate.

We envision the use of a combination of multi-hop wireless mesh networks and the existing Internet to support real-time communication between the avatars and human supervisor. However, the need for reliable and real-time communication imposes several significant challenges. Specifically, the current Internet architecture cannot guarantee any end-to-end QoS requirement for time-critical control flows and most existing

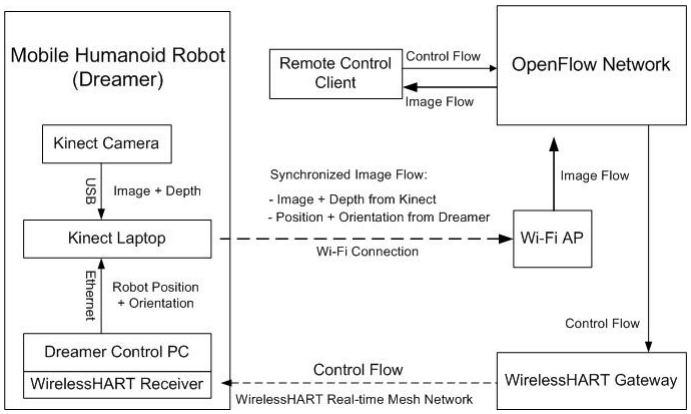


Fig. 5: The R^3 communication infrastructure for cyberphysical avatars

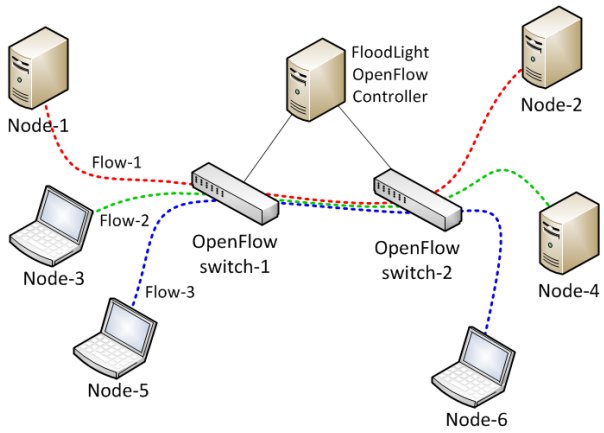


Fig. 6: OpenFlow testbed overview

wireless standards and routing protocols are not designed with real-time delay constraint in mind and as a result cannot provide any bound on end-to-end delay. Moreover, the inherent lossy wireless medium, the constantly fluctuating traffic volumes and channel conditions, together with complicated interference relationship have made it challenging to achieve high end-to-end reliability, which is essential for remote avatar-human communication.

A cyberphysical avatar typically contains two types of data flows. There is one or multiple data flows destinated to remote human supervisor containing physical information of the environment in which the avatar operates. These data flows include image flows captured by the cameras installed either on the avatar or in the operation environment, and real-time position/direction information of the avatar. There is also a control flow originated from the human operator to supervise the robot to execute designated tasks. Figure 5 presents an overview of our remote, reliable and real-time (R^3) communication infrastructure. We use Wi-Fi connection to transmit data flows because they require larger bandwidth but have soft real-time requirements on packet delivery. On the other hand, the time-critical control flow is transmitted on WirelessHART [19] real-time mesh network which is set up at the edge of the communication infrastructure to guarantee the end-to-end delay in the avatar-human communication. OpenFlow [20] network is deployed to enhance the existing Internet architecture to provide guaranteed QoS support. Our communication infrastructure has the following three key components.

A. *Wi-Fi connection for supporting data flows*

We use Wi-Fi connection at the edge of the communication infrastructure to help forward data flows to the remote human supervisor. In our current setting, the data flows are image streams captured by the Kinect sensor installed on the avatar, and the IP camera installed in its operation environment. Since the control PC on the avatar is battery-powered and its computation workload is intensive, the Kinect sensor is attached to a separated Laptop (Kinect Laptop). The Kinect Laptop and the avatar control PC are connected through Ethernet.

We utilize open source software library OpenKinect [21] to control the Kinect sensor. Both the color images and depth images received from the Kinect sensor will be synchronized with the position and orientation information received from the avatar control PC and sent to the remote supervisor. The remote control application receives the images and renders them on the user interface. It allows the operator to monitor the physical environment the avatar is operating in and supervise it to execute designated tasks.

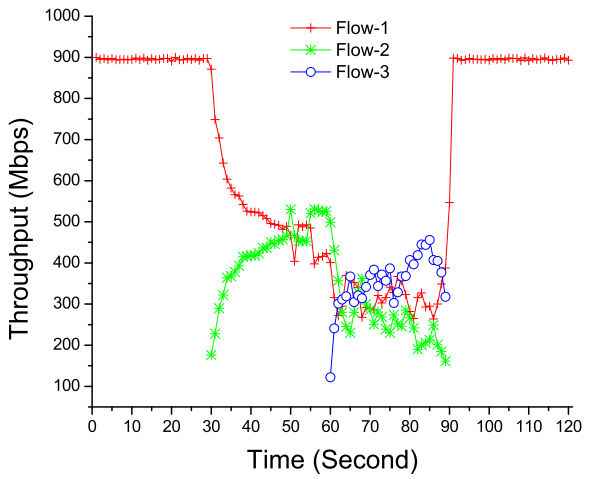
B. *OpenFlow network for providing QoS guarantees*

Cyberphysical avatar technology is a highly interactive application that requires strict timing constraint on the control flows and guaranteed bandwidth for the data flows. Building such communication infrastructure is challenging because the current Internet architecture has no facility to guarantee minimum bandwidth and end-to-end latency for network flows. To address this problem, we enhance the existing Internet architecture by deploying OpenFlow [20] network to connect the remote human supervisor and the avatar operation environment. OpenFlow is an open standard that provide QoS support and enables researchers to run experimental protocols in campus networks. To deploy an OpenFlow network, it requires two key components, the interconnected OpenFlow switches and the OpenFlow controller.

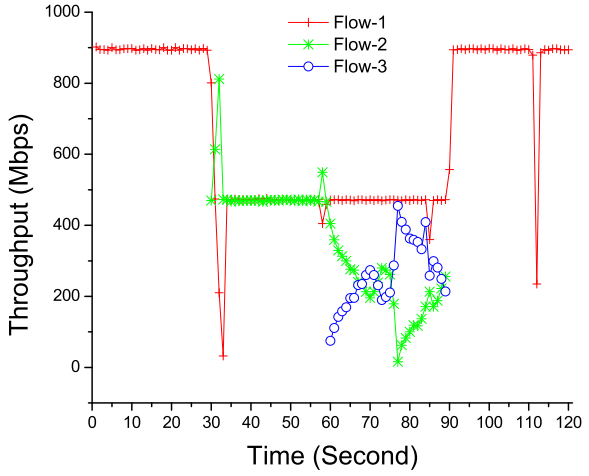
The hardware of OpenFlow switch is similar to current Ethernet switch, where we enhance the switch to support OpenFlow protocol. OpenFlow protocol provides an interface for remote controller to program the data plane of switches, which determine the corresponding action of network flows. These actions could be forwarding a packet to specific port, dropping out a packet, or putting it to a queue. By manipulating the performed actions on the switches, we can provide QoS guarantee to network flows. OpenFlow controller is usually a PC, where we can specify high level QoS requirement, then it can interpret it into low level actions and program these actions through OpenFlow protocol into OpenFlow switches.

To achieve end-to-end latency requirement for the control flow, we enhance the OpenFlow switch to support Virtual Clock Server [22]. By using Virtual Clock Server, we can bound the queuing time that a packet goes through a switch. By bounding the delay of each switch, we can calculate and provide the end-to-end delay bound [23] from human supervisor to the avatar.

In order to demonstrate the minimum bandwidth guarantee provided by OpenFlow switches, we setup a testbed as shown in Fig. 6. In the testbed, we deploy two switches, switch-1 (Pronto 3290) and switch-2 (Pronto 3295), and we enhance them with Indigo [24] open source OpenFlow firmware implementation. An OpenFlow controller PC running FloodLight OpenFlow controller [25] is used to configure the OpenFlow switches. We connect three PCs/Laptops to switch-1 to serve as clients, and three other PCs/Laptops are connected to switch-2 serving as servers. We run iperf to generate TCP flows with maximum



(a) Throughput of three flows without rate guarantee



(b) Throughput of three flows with rate guarantee

Fig. 7: Demonstration of bandwidth guarantee in OpenFlow switch

rate. Each computer in this experiment is equipped with gigabit Ethernet network interface, and the maximum achievable throughput of each flow is around 900Mbps if no other traffic is present in this network.

We configure three flows in the testbed and evaluate their throughput with and without rate guarantee. In the baseline experiment without rate guarantee, we first start flow-1 at time 0, and we run flow-1 for 120 seconds. Flow-2 is started at time 30, and its duration is 60 seconds. Flow-3 starts at time 60 and runs for 30 seconds. In the second experiment with rate guarantee, we reserve 500Mbps bandwidth for flow-1 by setting a minimum rate guarantee queue at the egress port of switch-1. We then run the same setting as the baseline experiment.

Fig. 7 summarizes our experiment results. Fig. 7a shows the throughput of the three flows in the baseline experiment. Without rate guarantee, all three flows compete the link capacity of the connection link between switch-1 and switch-2. When three flows are present at the same time (from time 60 to 90), the throughput of each flow is around 300Mbps. In the second experiment as shown in Fig. 7b, because we enable the minimum rate guarantee queue for flow-1, the throughput of flow-1 can be maintained around 500Mbps, even though in present of the other two flows.

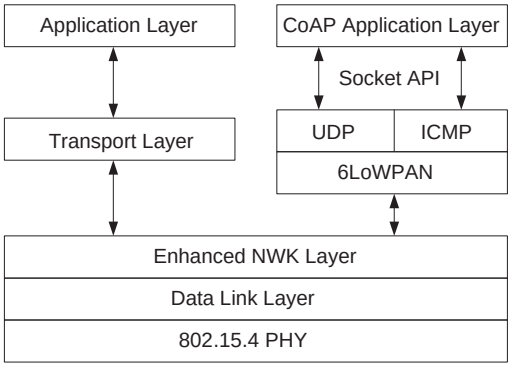


Fig. 8: Design of the WirelessHART real-time communication stack

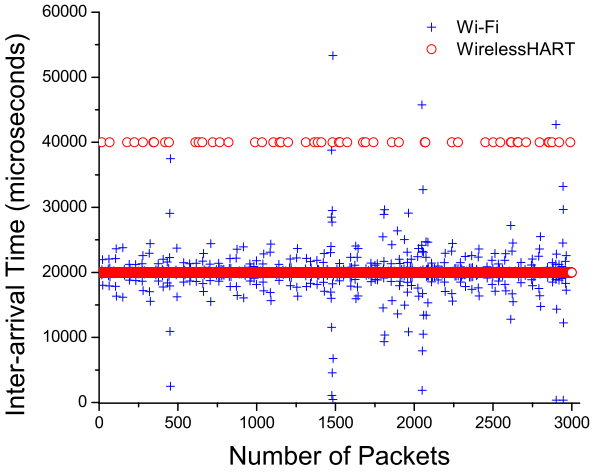
C. *WirelessHART mesh for supporting real-time control flow*

The control flow from the remote human supervisor to the robot control PC is time-critical and has hard deadline on its delivery. Due to the pervasive Wi-Fi signals and the backoff mechanism used in 802.11, the jitter in Wi-Fi transmission is large and unpredictable. This is a fatal disadvantage of Wi-Fi to be adopted for providing reliable and real-time communication for the control data flow in cyberphysical avatars.

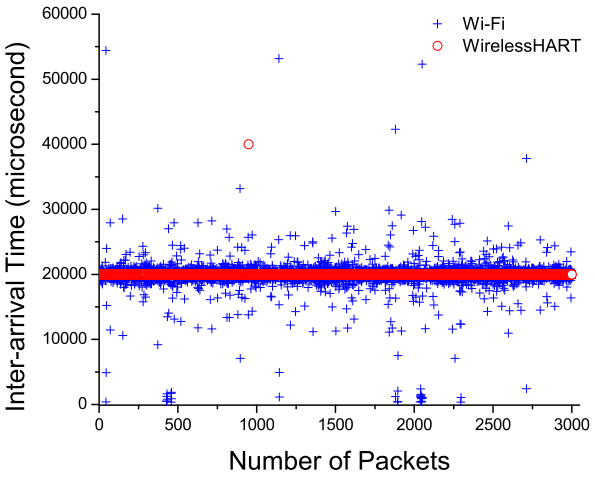
For this reason, in the R^3 communication infrastructure, we use WirelessHART [26] real-time mesh network to transmit the control data flow to the avatar control PC. The control flow is transmitted from the remote control application to the WirelessHART Gateway through the deployed OpenFlow network and then further relayed to the avatar control PC on WirelessHART mesh.

Figure 8 presents the architecture of the WirelessHART real-time communication stack. At the bottom, it adopts IEEE 802.15.4-2006 [18] as the physical layer for energy-saving purpose. On top of that, WirelessHART defines a TDMA-based data link layer. It provides strict 10 ms timeslot and network-wide time synchronization to support deterministic packet delivery. Retransmission, channel hopping and channel blacklisting mechanisms are also applied to improve the communication reliability. The network layer supports self-organizing and self-healing mesh networking techniques to further improve the network performance in noisy and harsh environments. To improve the service scalability and make the application development easier, we further enhance the communication stack with a 6LoWPAN adaptation layer, a UDP transportation layer and a CoAP application layer. This enhancement makes the WirelessHART device IP-enabled and the Gateway only needs to take the role of the router and forward the IP packets to the avatar control PC. Only the remote control application and the application running on avatar control PC need to understand the specific application protocol. The Gateway can remain unchanged when new services are established between the supervisor and the robot.

To demonstrate the benefit of WirelessHART real-time protocol, we conduct a set of experiments to compare the packets inter-arrival time (IAT) between Wi-Fi and WirelessHART. Both the Wi-Fi and WirelessHART network are deployed in the graduate student office at UT ACES 5th floor. We configure both the Wi-Fi and WirelessHART devices to periodically publish data every 20*ms* in the media access control (MAC) layer, and we calculate the IAT on the receiver side. For each network, we run the experiment independently for 60 seconds and our results



(a) Without present of jammer



(b) In present of jammer

Fig. 9: Inter-arrival time comparison between Wi-Fi and WirelessHART in office environment

are summarized in Fig. 9.

Fig. 9a shows the IAT comparison in regular office environment. Because Wi-Fi utilizes CSMA-CA and random backoff mechanisms to coordinate channel access, it cannot transmit packets in a deterministic way, thus has high variation at the packet transmission time. WirelessHART on the other hand adopts TDMA mechanism and only transmits packets at fixed time points periodically. We observed from Fig. 9a that most WirelessHART packets are transmitted exactly every 20ms and only 1.7% of WirelessHART packets are retransmitted once due to interference from other Wi-Fi traffic, and have doubled IAT.

Fig. 9b shows the behavior of WirelessHART and Wi-Fi networks in present of intended interference. The jamming signal is created by deploying another Wi-Fi network, where we disable the carrier sense and random back mechanisms of its sender and use iperf to create maximum Wi-Fi traffic. Because of the interference from the jamming signal, we observed higher IAT variation in the Wi-Fi network. In WirelessHART network, we enable the dynamic channel blacklisting mechanism to mask out ill-conditioned channel. By hopping to less interference channel, most of WirelessHART packets are successfully transmitted without retransmission.

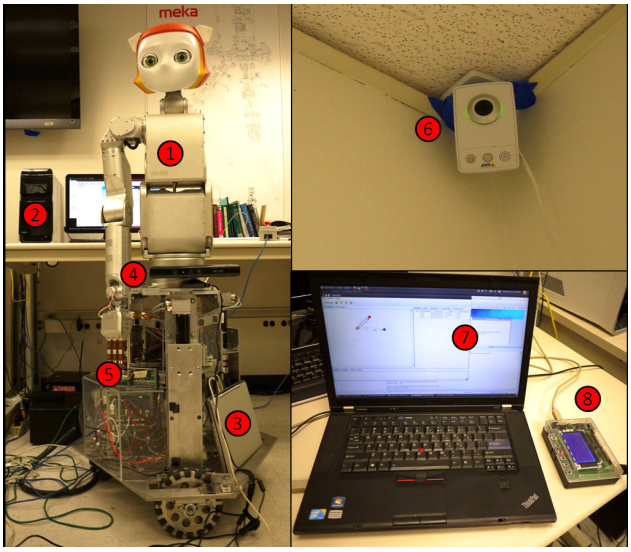


Fig. 10: An overview of the system setup in UT Human Centered Robotics Lab. (1) The Dreamer robot. (2) Robot control PC. (3) Kinect Laptop. (4) Kinect sensor. (5) WirelessHART receiver. (6) IP camera. (7) WirelessHART Gateway. (8) WirelessHART Access Point.

V. Designing and building a cyberphysical avatar

We have completed a prototype of the cyberphysical avatar to verify the effectiveness of the proposed architecture. The remote control application is installed in UT ACES building and the robotic system is located in UT Human Centered Robotics Lab. OpenFlow switches are being deployed in UT campus network to provide bandwidth guarantee for reliable and real-time avatar-human communication. In this section, we will present the details of the system design and integration. A video demo of remote supervision on the avatar to execute "touch" and "incremental move" commands is available online [27].

A. System Integration in Human Centered Robotics Lab

Figure 10 presents an overview of the system setup in the Human Centered Robotics Lab. The control flow originated from the remote operator goes through the OpenFlow campus network and is transmitted to the Dreamer robot through the local real-time wireless communication subsystem. The Kinect camera and IP camera installed in the lab keep track of the robot's motion and its operation environment by sending image flows back to the remote supervisor. Based on these image flows, the remote human operator can supervise the robot to execute designated tasks by sending appropriate commands. Our system has the following three key components.

The Dreamer/Meka Hardware: The main hardware tool that we use for this study is the Dreamer/Meka mobile dexterous humanoid robot. This robot includes the T2 Meka torso, the A2 Series Elastic Meka arm, the H2 tendon driven Meka hand, the Dreamer/Meka head co-developed by Meka and UT Austin, and the torque-controlled holonomic UT Austin's Tricky base. The actuators for the base and the upper body, except for the head, contain torque/force sensors that enable Elmo amplifiers to implement current or torque feedback. An Ethercat serial bus communicates with sensors and motor amplifiers from a single computer system. A PC running Ubuntu Linux with the RTAI Realtime Kernel runs the models and control infrastructure described in Section II. The Tricky holonomic base contains torque sensors as well as the inertial measurement unit (IMU)

3DM-GX3-25 from MicroStrain. It achieves holonomic motion and force capabilities by utilizing Omni wheels located in a equilateral triangular fashion. Interested readers are referred to [28] for more details about this robot.

Kinect/IP Cameras: We have two cameras installed in the Human Centered Robotics Lab. An IP camera is installed at the right upper corner to give an overview of the operation environment of the Dreamer robot; A Kinect camera is installed in front of the robot to capture the image and depth information of the target. As we mentioned in Section IV, due to the limitation on the power and the computation capability, the Kinect camera is installed on a separate Laptop (Kinect Laptop in Figure 10). The Kinect Laptop is connected to the avatar controller through Ethernet. It synchronizes image streams (including the image and depth information) captured from the Kinect camera and the position/orientation information of the robot together and sends to the remote control application.

Real-time wireless communication subsystem: We set up a WirelessHART network for achieving real-time wireless communication at the edge of the communication infrastructure. The communication subsystem includes a WirelessHART Gateway which is connected to the UT campus network, and a WirelessHART receiver which is connected to the avatar control PC to provide real-time communication. More intermediate devices can be deployed to form a mesh to cover larger area if necessary. The WirelessHART receiver exchanges control commands and robot status with the avatar control PC through shared memory. The robot status information is transmitted back to the remote operator on the WirelessHART real-time wireless network in the reversed direction. To provide deterministic communication, we establish a superframe with the size of 10 timeslots and create 5 pairs of transmit/receive links between the WirelessHART receiver and Gateway, so the frequency of the control flow can be up to 50Hz, which is sufficient for high-level control command transmission. As the ongoing work, we are enhancing the 802.11 standard with real-time and reliable features. We target at building a general wireless platform to support a wide range of wireless sensing and control applications by achieving a good balance among sampling rate, reliability, and energy efficiency.

B. Remote Control Application

Figure 11 shows a screen capture of the remote control application we developed for supervising the Dreamer robot. The details of the interface messages between the control application and the Kinect Laptop/Dreamer robot can be found in the technical report [8].

The specific skill we are training the Dreamer robot is to move itself close to a desk and pick up a designated target under supervision. As shown in Fig. 11, in the remote control UI, the color images and the depth images from Kinect camera and the images from the IP camera are displayed in the three image panels at the top of the UI. The human supervisor can choose a target in the color image from Kinect panel by clicking on it. After clicking on the target, a copy of the color image at this moment is copied to the image of target object panel. A red dot is added on the image showing the position the user clicked. The coordinate of this position is also displayed in the mouse clicked position label. Using the position of the click on the image and the depth data at that point, we calculate the physical

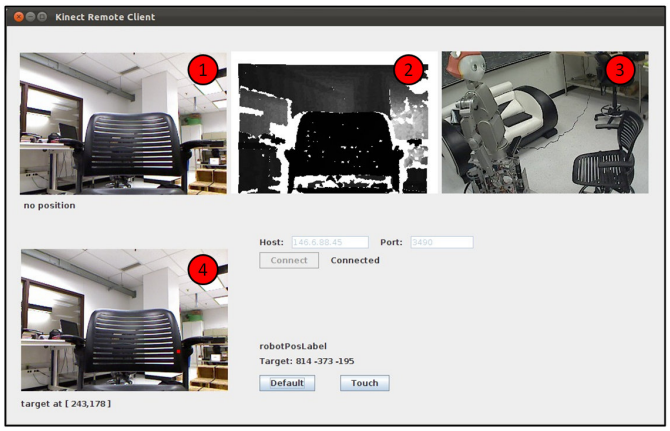


Fig. 11: A screen capture of the remote control application. (1)(2) Color and depth image from Kinect sensor. (3) Image from IP camera. (4) Image snapshot when user presses the color image.

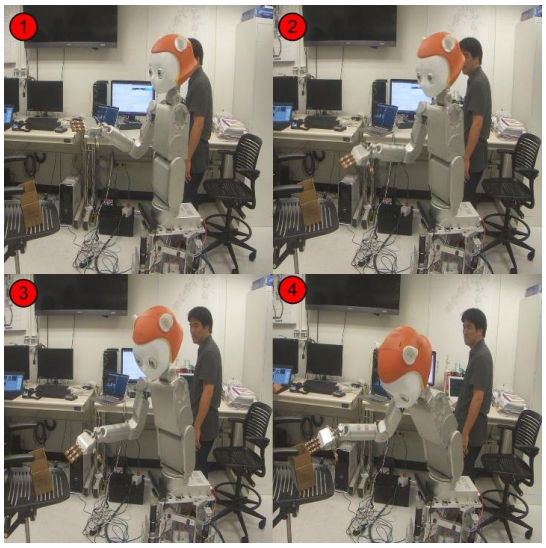


Fig. 12: Video snapshots for Touch and Incremental Move commands.

coordinate of the target with respect to the Kinect. Once get the physical coordinate of the target, user can issue commands to the Dreamer robot to execute specific tasks. In our current testbed, three commands have been implemented already: Default, Touch and Incremental Move. The Default command sets the robot back to its default gesture, while the Touch command asks the robot to touch the target the user clicked on. Fig. 12 is a sequence of video snapshots that demonstrate how the robot reacts when a Touch command is received. In the first video frame, the robot hand started from the default position. The robot moved to the designated position as shown in frame 2 and 3. It is possible that the Kinect sensor has small measurement error due to its hardware limitation. In that case, we relied on visual feedback, and used Incremental Move command to direct the robot to touch the target object as shown in frame 4. The grasp skill to lift up the target is still under training and the Grasp command will be added to the remote control application as soon as the training is finished.

VI. Conclusion and Future works

This paper introduces the concept of a cyberphysical avatar which is defined to be a semi-autonomous robotic system that adjusts to an unstructured environment and performs physical tasks subject to critical timing constraints while under human

supervision. A cyberphysical avatar is the bridge technology that will help transition dumb teleoperated robotic devices to autonomous robots capable of functioning in unstructured environments. What makes the cyberphysical avatar possible today is the convergence of three recent technologies: body-compliant control in robotics, neuroevolution in machine learning and QoS guarantee in real-time communication. By integrating these technologies, we have built a prototype cyberphysical avatar testbed. Building this physical testbed is an essential first step for exploring the cyberphysical avatar concept because the physical and computational complexities involved necessarily require less than exact modeling and the use of mathematical approximations to simulate physical processes; the viability of the cyberphysical avatar must be validated by a physical testbed. This paper describes the implementation of the major components of the cyberphysical avatar and some of their performance results. The cyberphysical avatar is fully operational and is being trained to acquire some basic physical skills by the NEAT [7] algorithm invented by one of the coauthors.

The availability of the cyberphysical avatar testbed opens up new avenues for future research that arise from the interaction between robotics, machine learning and real-time system design. We mention below a couple of the issues to be explored.

- Models for real-time resource allocation and scheduling must be explored that are neither hard nor soft real-time in that a cyberphysical avatar is best characterized as a hierarchical, "contract-driven" real-time system. For example, the avatar may be commanded by the human supervisor to trade off walking speed against visual processing accuracy of its environment; in an uncluttered environment, the avatar can move faster without worrying about being tripped over by an unexpected obstacle. This will involve trading off the quality of the sensor data for faster response time in the gait-maintenance control loop, while keeping the related mode change overhead low. This calls for a proactive approach in providing performance guarantees.
- New concepts of compositionality are needed to make it easier to compose physical components into higher-level semantic units. For example, in compliant control, postures are objectives that need to optimize a performance objective in the null space of tasks (which are coordinates to track a position or force trajectory). Since attaining desired postures requires non-trivial computation in real time, fast tests for checking feasibility will be very useful for helping the human supervisor to make decisions. This might be possible if we can lift the level of abstraction of the compliance problem to a mixed logical-numerical constraint satisfaction formulation and exploit techniques in run-time verification.
- Experimental work is required to qualify and quantify the efficacy of human supervision of the cyberphysical avatar in performing specific tasks. For example, what level of supervision and at what computational and communication costs can a cyberphysical avatar and human supervisor team put together, say, an Ikea furniture set?

  These and other issues will require many months if not years of work in the future.

References

[1] Song Han, Aloysius K. Mok, Jianyong Meng, Yi-Hung Wei, Pei-Chi Huang, Xiuming Zhu, Luis Sentis, Kwan Suk Kim, Risto Miikkulainen, and Jacob Menashe, "Architecture of a cyberphysical avatar," *to appear in International Workshop on Real-Time and Distributed Computing in Emerging Applications (REACTION 2012)*.

[2] L. Sentis, *Motion Planning for Humanoid robots*, chapter Compliant Control of Whole-Body Multi-Contact Behaviors in Humanoid Robots, pp. 29–63, Springer Berlin Heidelberg, 2010.

[3] L. Sentis, J. Park, and O. Khatib, "Compliant control of multi-contact and center of mass behaviors in humanoid robots," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 483–501, June 2010.

[4] Risto Miikkulainen, "Neuroevolution," in *Encyclopedia of Machine Learning*. 2010.

[5] Simon S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, Upper Saddle River, New Jersey, third edition, 2009.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[7] Kenneth O. Stanley and Risto Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[8] Song Han, Aloysius K. Mok, Jianyong Meng, Yi-Hung Wei, Xiuming Zhu, Luis Sentis, Kwan Suk Kim, Risto Miikkulainen, and Jacob Menashe, "Architecture of a cyberphysical avatar," *UTCS Technical Report #TR-12-12.* `http://apps.cs.utexas.edu/tech_reports/reports/tr/TR-2082.pdf`.

[9] Matei Ciocarlie, Columbia University, ," `http://sourceforge.net/projects/graspit/files/releases/`.

[10] Nick Jakobi, *Minimal Simulations for Evolutionary Robotics*, Ph.D. thesis, School of Cognitive and Computing Sciences, University of Sussex, 1998.

[11] Hod Lipson, Josh Bongard, Victor Zykov, and Evan Malone, "Evolutionary robotics for legged machines: From simulation to physical reality," in *Proceedings of the 9th International Conference on Intelligent Autonomous Systems.*, 2006, pp. 11–18.

[12] Faustino Gomez and Risto Miikkulainen, "Transfer of neuroevolved controllers in unstable domains," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Berlin, 2004, Springer.

[13] Vinod Valsalam, *Utilizing Symmetry in Evolutionary Design*, Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin, 2010, Technical Report AI-10-04.

[14] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA, 2010, pp. 119–126, ACM.

[15] Orazio Miglino, Henrik Hautop Lund, and Stefano Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial Life*, vol. 2, pp. 417–434, 1995.

[16] Juan Cristóbal Zagal and Javier Ruiz-Del-Solar, "Combining simulation and reality in evolutionary robotics," *Journal of Intelligent and Robotic Systems*, vol. 50, pp. 19–39, 2007.

[17] S. Bak, D.K. Chivukula, O. Adekunle, Mu Sun, M. Caccamo, and Lui Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, april 2009, pp. 99 –107.

[18] "IEEE 802.15.4 WPAN Task Group," `www.ieee802.org/15/pub/TG4.html`.

[19] "WirelessHART," `http://www.hartcomm.org/protocol/wihart/wireless_technology.html`.

[20] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, 2008.

[21] "Openkinect," `www.openkinect.org`.

[22] Geoffrey G. Xie and Simon S. Lam, "Delay guarantee of virtual clock server," *IEEE/ACM Trans. Netw.*, 1995.

[23] Pawan Goyal, Simon Lam, and Harrick Vin, "Determining end-to-end delay bounds in heterogeneous networks," *Network and Operating Systems Support for Digital Audio and Video Springer Berlin / Heidelberg*, vol. 1018, pp. 273–284, 1995.

[24] "Indigo open source firmware for openflow swiches," `http://www.openflowhub.org/display/Indigo/`.

[25] "Floodlight openflow controller," `http://floodlight.openflowhub.org/`.

[26] Jianping Song, Song Han, Al Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt, "WirelessHART: Applying wireless technology in real-time industrial process control," in *Proc. of IEEE RTAS*, 2008.

[27] "Avatar Demo," `http://www.cs.utexas.edu/~shan/avatar.html`.

[28] F. Lima J.G. Petersen K.S. Kim L. Sentis P.D. Wong, S. Gupta, "Building an educational whole-body compliant mobile humanoid robot for safe rough-terrain physical hri," in *IEEE International Conference on Intelligent Robots and Systems (under submission)*, 2012.