Intel Serv Robotics (2016) 9:31–48
DOI 10.1007/s11370-015-0181-2

ORIGINAL RESEARCH PAPER

Optimization of humanoid's motions under multiple constraints in vehicle ingress task

Kiwon Sohn[1] · **Paul Oh**[1]

Received: 24 December 2014 / Accepted: 17 July 2015 / Published online: 26 July 2015

Abstract This paper presents an approach on whole-body motion optimization for a humanoid robot to enter a ground vehicle. Motion capture system (mocap) was used to plan an initial suboptimal motion. Reinforcement learning was then implemented to optimize the trajectories with respect to kinematic and torque limits at the both body and the joint level. The cost functions in the body level calculated a robot's static balancing ability, collisions and validity of the end-effector movement. Balancing and collision checks were computed from kinematic models of the robot and the vehicle model. Energy consumption such as torque limit obedience was checked at the joint level. Energy cost was approximated as joint torque, measured from a dynamic model. Various penalties such as joint angle and velocity limits were also computed in the joint level. Physical limits of each joint ensured both spatial and temporal smoothness of the generated trajectories. Finally, experimental evaluations of the presented approach were demonstrated through simulation and physical platforms in a real environment.

Keywords Humanoids · Vehicle mounting · Whole body motions generation · Optimal trajectory planning · Hubo+

Electronic supplementary material The online version of this article (doi:10.1007/s11370-015-0181-2) contains supplementary material, which is available to authorized users.

✉ Kiwon Sohn
skw1125@gmail.com

Paul Oh
paul.oh@unlv.edu

1 Mechanical Engineering Department, University of Nevada, Las Vegas, NV 89154, USA

1 Introduction

Enabling humanoids to *drive* a ground vehicle differs from the development of driverless cars. Efforts like the DARPA Challenges [1] or the Google Car [2] focus on engineering the vehicle. By contrast, there is merit and broader impacts if humanoids can drive off-the-shelf *unmodified* cars. A big picture is for future robots to operate human tools and also locomote in human-centered environments. Vehicle driving can be an example; the car is a tool which people use to get around and its cockpit is human-centered. Today's humanoids, however, rarely have the range-of-motion, balance control and motion planning algorithms to get inside the car by themselves (ingress). Furthermore, such humanoids often do not have the perception and cognition to drive a vehicle. Therefore, programming a humanoid for vehicle handling (ingress, drive, and egress) has intellectual merit.

This paper specifically describes humanoid whole-body motion plans to ingress a doorless vehicle (Fig. 1). Ingress is the first step towards the larger goal of humanoid vehicle handling. Ingressing is difficult and an open research area demanding perception, balancing, and handling both external and self collisions. Observations of the elderly or the physically impaired perhaps underscore these difficulties; those with limited range-of-motion must carefully ingress by constraining limb positions. Similarly robots with limbs of limited force and torque may also have similar motion strategy. As such, ingress requires coordinating a robot's many degree-of-freedom (DOF) and planning motions respect to multiple time-varying constraints.

Robots, especially humanoids, involve large search spaces due to their high DOF structures. To plan motions, search-based sampling approaches like Rapidly Exploring Random Trees (RRT) [4] and its variants IKBiRRT [5] and CBiRRT [6] are often used. These search-based algorithms are well

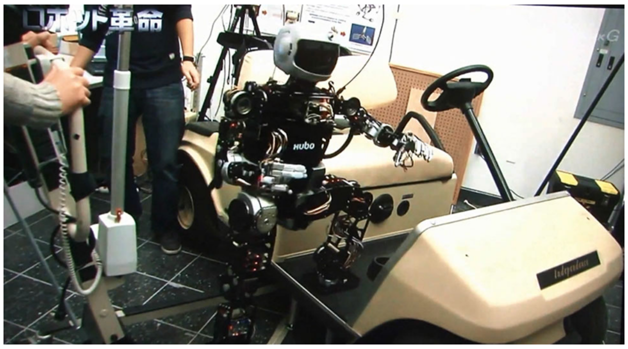


Fig. 1 Humanoid ingressing golf cart (Club Car DS IQ) [3]

suited for a humanoid structure because the its end-effector can serve as the end-goal [7] in search spaces. These methods can generate a path in the humanoid's configuration space and can guarantee collision avoidance and static balance. However, these algorithms do not consider factors like torque and energy efficiency which are important for humanoid's safe ingress. Furthermore, these algorithms rarely yield robot motions that mimic human ones. It is also difficult to check the robot's static balance, especially during foot-switching while ingressing.

Another more direct approach for complex motion planning is to employ captured human motion data. Such approaches are traditionally used in animation [8], but have also been applied to generate humanoid's whole-body motions [9,10]. The captured data can be used to produce human-like motions but such movements are rarely optimized to be energy efficient. The optimal motions for a person and a robot often differ; the multitude of muscles in the human body often invoke different joints. Another limitation of using pre-recorded human motions is that factors like collision and static balance are not considered in those planning approaches. However, ingress requires such factors to be considered.

As such, initial planned motions from path planning algorithms should be optimized with respect to various time-varying constraints. Recently, a broad range of optimization techniques also have been applied to humanoid motion planning. For instance, [11] and [12] adopt Lagrange multiplier to limit planned motions within joint constraints. They re-express the unconstrained optimization problems by non-linear least square (NLLS) model to include all defined constraints. After converting the NLLS into linear model by Gauss–Newton method, the optimization techniques generated a final trajectory by gradient descent updates. This process generates a trajectory which is the closest to the best solution under the defined constraints. The problem, however, is that such optimization minimizes an error which is the sum of costs from all the different constraints. Therefore, other candidate trajectories could exist for different static and dynamic features. The errors in these candidate trajectories may be slightly bigger than the one for the optimal solution. However, these optimization techniques provides only the *single* best trajectory under the given constraints.

This paper introduces another way of planning and optimizing ingress motions for humanoids. Proposed is a reinforcement learning agent-based optimization method. Vehicle ingress demands optimizing planned trajectories with various time-varying constraints. To meet this, in the presented approach, the learning agent chooses an optimal sequence of motion states from choices with respect to given kinematic and dynamic constraints. This learning-based method also exploits multiple solutions under defined constraints with the graded state-action value table. For experimental verification of the presented approach, the optimized trajectory was tested and evaluated both in simulation and in experiments with Hubo.[1] The net result is that the proposed approach combines several well-defined existent techniques (such as Q-learning, mocap, RRT and ProPac) to solve the challenging motion of vehicle ingress in a different way from other humanoid trajectory optimization techniques that often adopt Lagrange multiplier or linear quadratic methods [13].

Section 2 describes an overview of an optimization framework using a reinforcement learning agent. Section 3 demonstrates a guideline path design using a motion capture system (mocap) and post-processing. Sections 4 and 5 show cost functions which define kinematic and dynamic constraints of the humanoid in body and joint levels respectively. Section 6 presents final output trajectories design and a computational complexity of the presented approach. Section 7 demonstrates experimental results with optimized trajectories in test-and-evaluation stage and presents future studies. Section 8 concludes with future work.

2 Trajectory-optimization framework

2.1 Static motion

Enabling a full-sized humanoid to ingress an utility vehicle is not trivial. There are many unknown factors for vehicles, such as, suspension level or tire air pressure. Such factors are difficult to model accurately. Also, each tire and shock absorber for the vehicle may differ. Therefore, planning dynamic motion is not often realistic in designing of ingress motion. Motion planning should be updated statically to be stable enough to offset unknown dynamic factors. It is critical for the static motion to have the robot's discrete con-

[1] Specifically the Hubo+ humanoid (released in 2010) was used. Hubo+ is the generation following the 2007 KHR-4 Hubo. The generic term "Hubo" is used to refer to the humanoid used in this paper.

figurations which satisfy static constraints on pre-determined time frames. Each selected time frame can be determined by its importance in performance, such as foot-switching or center-of-mass (CoM) changes.

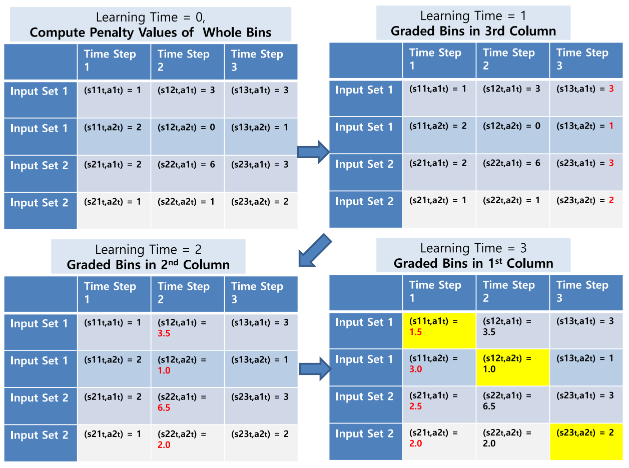
Such key frames can be generated with discrete time steps based on output frames from various path planning algorithms. However, discrete frames which are not enough dense can make critical issues such as collisions or torque limit obedience in generated trajectories. In this study, a 100 ms time step was used for getting key frames from an initial path-planned trajectory. The value was chosen based on the maximum value which does not violate various time-varying constraints such as collision avoidance and joint limits. Section 6.2 describes more details of determining a sampling frequency for key frames in the proposed framework.

2.2 Trajectory-optimization with reinforcement-learning (inverse grading of Q value)

Though sampling method or mocap can generate input key frames for humanoids, they often do not meet important constraints which should be considered for humanoid vehicle mounting.[2] Search-based sampling approaches [4–6] do not consider energy efficiencies or static balance during foot-switching. Mocap-based approaches [8–10] often do not account for factors such as collisions, static balance and kinematic differences.

Therefore, key frames using any path planning algorithms should be optimized with respect to all time-varying constraints. This paper presents a humanoid motion optimization framework for vehicle mounting. The optimizing process follows this sequence: (1) guideline key frame trajectories are designed initially using a selected path planner; (2) key frames are then modified to meet multiple required static constraints; (3) a humanoid's smooth joint angle trajectories are generated as a final output. The presented framework has a limited number of states (i.e., key frames) which should be optimized with constraints. In this study, reinforcement learning is applied to key frames to meet multiple time-varying constraints. A reinforcement learning agent can interact with its feedback values from penalty cost functions and grade states efficiently in discrete time steps [14, 15].

For the reinforcement learning agent, a Q algorithm is used [16]. The trajectory used in the framework has a finite set of states (key frames) over discrete time steps. A transition between one state and any of the states in the next time step is defined as a possible action for the state. Therefore, there are finite sets of actions in a learning agent. In sum, states exist in discrete time steps and the decision maker



Learning Time = 0,
Compute Penalty Values of Whole Bins

	Time Step 1	Time Step 2	Time Step 3
Input Set 1	$(s11_t,a1_t) = 1$	$(s12_t,a1_t) = 3$	$(s13_t,a1_t) = 3$
Input Set 1	$(s11_t,a2_t) = 2$	$(s12_t,a2_t) = 0$	$(s13_t,a2_t) = 1$
Input Set 2	$(s21_t,a1_t) = 2$	$(s22_t,a1_t) = 6$	$(s23_t,a1_t) = 3$
Input Set 2	$(s21_t,a2_t) = 1$	$(s22_t,a2_t) = 1$	$(s23_t,a2_t) = 2$

Learning Time = 1
Graded Bins in 3rd Column

	Time Step 1	Time Step 2	Time Step 3
Input Set 1	$(s11_t,a1_t) = 1$	$(s12_t,a1_t) = 3$	$(s13_t,a1_t) = 3$
Input Set 1	$(s11_t,a2_t) = 2$	$(s12_t,a2_t) = 0$	$(s13_t,a2_t) = 1$
Input Set 2	$(s21_t,a1_t) = 2$	$(s22_t,a1_t) = 6$	$(s23_t,a1_t) = 3$
Input Set 2	$(s21_t,a2_t) = 1$	$(s22_t,a2_t) = 1$	$(s23_t,a2_t) = 2$

Learning Time = 2
Graded Bins in 2nd Column

	Time Step 1	Time Step 2	Time Step 3
Input Set 1	$(s11_t,a1_t) = 1$	$(s12_t,a1_t) = 3.5$	$(s13_t,a1_t) = 3$
Input Set 1	$(s11_t,a2_t) = 2$	$(s12_t,a2_t) = 1.0$	$(s13_t,a2_t) = 1$
Input Set 2	$(s21_t,a1_t) = 2$	$(s22_t,a1_t) = 6.5$	$(s23_t,a1_t) = 3$
Input Set 2	$(s21_t,a2_t) = 1$	$(s22_t,a2_t) = 2.0$	$(s23_t,a2_t) = 2$

Learning Time = 3
Graded Bins in 1st Column

	Time Step 1	Time Step 2	Time Step 3
Input Set 1	$(s11_t,a1_t) = 1.5$	$(s12_t,a1_t) = 3.5$	$(s13_t,a1_t) = 3$
Input Set 1	$(s11_t,a2_t) = 3.0$	$(s12_t,a2_t) = 1.0$	$(s13_t,a2_t) = 1$
Input Set 2	$(s21_t,a1_t) = 2.5$	$(s22_t,a1_t) = 6.5$	$(s23_t,a1_t) = 3$
Input Set 2	$(s21_t,a2_t) = 2.0$	$(s22_t,a2_t) = 2.0$	$(s23_t,a2_t) = 2$

Fig. 2 Example of state-action value inverse grading with Q learning algorithm (one iteration)

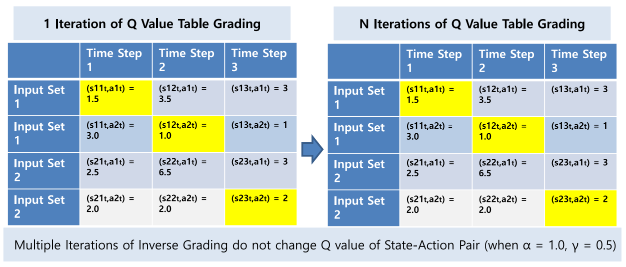
can choose any action that is available in each state. Therefore, it satisfies requirements of the Markov decision process (MDPs). Specifically, Q-learning can be used to find an optimal action-selection policy for the MDPs [17].

Figure 2 demonstrates a simple example of state-action value table (Q table) grading with Q learning algorithm. In the example, there are two input trajectories and they have three time steps. Each input trajectory generates individual states. Also an action is defined as one of any transitions between states in consecutive time steps. In this example, there are two states and each state has two actions per time frame. Therefore, four sets of state-and-action pairs are generated in each time step. Since states in the 1st and the 2nd row of the table are generated from the same input trajectory 1, they both are denoted as Input Set 1. Same rule applies for Input Set 2 in the 3rd and the 4th row of the table.

In this study, the Q value table has a limited number of states. Each state is a key frame of the pre-designed initial trajectory and each action is a transition between known states. Therefore, a penalty value of each state-and-action pair in Q table can be calculated based on kinematic and dynamic cost of the pair. Such cost value of each bin is invariant over the updates. In such deterministic environment (Q value table), α (learning rate) is optimal with value 1. To balance between current rewards and long term high rewards, γ (discount factor) is set to 0.5.

The bins in the last column of the table have the previously known states. Therefore, the bins in the table can be graded inversely from the last column to the first one. This inverse grading accelerated the grading time of the Q value table. Since all bins in the last column do not have future states, their Q values are determined only by the current cost values. Furthermore, cost values of each bin are invariant over the updates. Therefore, every bin in the preceding time step has an invariant minimum future Q value regardless of num-

[2] The paper uses the term "mounting" to refer to either ingressing or egressing. These two tasks are the same if the motion plans are treated statically. They differ if accelerations need to be considered.



1 Iteration of Q Value Table Grading

	Time Step 1	Time Step 2	Time Step 3
Input Set 1	$(s11_t,a1_t)$ = 1.5	$(s12_t,a1_t)$ = 3.5	$(s13_t,a1_t)$ = 3
Input Set 1	$(s11_t,a2_t)$ = 3.0	$(s12_t,a2_t)$ = 1.0	$(s13_t,a2_t)$ = 1
Input Set 2	$(s21_t,a1_t)$ = 2.5	$(s22_t,a1_t)$ = 6.5	$(s23_t,a1_t)$ = 3
Input Set 2	$(s21_t,a2_t)$ = 2.0	$(s22_t,a2_t)$ = 2.0	$(s23_t,a2_t)$ = 2

N Iterations of Q Value Table Grading

	Time Step 1	Time Step 2	Time Step 3
Input Set 1	$(s11_t,a1_t)$ = 1.5	$(s12_t,a1_t)$ = 3.5	$(s13_t,a1_t)$ = 3
Input Set 1	$(s11_t,a2_t)$ = 3.0	$(s12_t,a2_t)$ = 1.0	$(s13_t,a2_t)$ = 1
Input Set 2	$(s21_t,a1_t)$ = 2.5	$(s22_t,a1_t)$ = 6.5	$(s23_t,a1_t)$ = 3
Input Set 2	$(s21_t,a2_t)$ = 2.0	$(s22_t,a2_t)$ = 2.0	$(s23_t,a2_t)$ = 2

Multiple Iterations of Inverse Grading do not change Q value of State-Action Pair (when α = 1.0, γ = 0.5)

Fig. 3 Iterations of state-action value inverse grading

ber of updates. In other words, future reward factors can not diverge the Q values. The net result is that only one iteration (update) of grading is required for the Q value table to get the optimized trajectory. This acceleration of convergences was possible since the all bins in the last column of the table has previously known states and actions. Figure 2 demonstrates such inverse grading process of the Q value table. After grading, searching and marking process which has the minimum Q value is iterated. The final output trajectory is thus the optimal sequence of state-action pairs generated by collecting marked states ($S_{11} - S_{12} - S_{23}$). Figure 3 shows that inverse grading does not diverge Q values with multiple iterations. Previous studies [18] demonstrate more details of grading process for a Q value table with given input trajectories.

Trajectory optimization with reinforcement learning also conveyed advantages over other trajectory optimization techniques. Many optimization techniques, such as Lagrange multiplier [11,12] or linear quadratic methods [13], provide only the *single* best trajectory under the given constraints. By contrast, reinforcement learning searches over many possible state-action pairs at *each* time step. Furthermore reinforcement learning records every cost value in the state-action table under given constraints. As such, penalty values of all the bins in the state-action value table can then be re-used as needed. Such re-use allows multiple solution trajectories that have below-threshold costs to be simply produced at any time. For example, in Fig. 2, if the cost threshold is set to 4, one more solution trajectory can be found additionally ($S_{21} - S_{12} - S_{23}$).

In this section, how a Q value table becomes graded with the value iteration process was presented. Q value of each state-and-action pair in the table was originally 0 and updated with corresponding kinematic and dynamic penalty costs of the pair. In this study, penalty values were chosen and defined based on the time-varying constraints of ingress. Sections 4 and 5 describes all the defined penalty costs in a robot's body and joint level each. Section 2.4 presents more details with how those penalty costs were integrated in the proposed trajectory optimization framework.

The net effect is that various cost functions are added to generate Q value of each bin in Q value table. Based on the relative importance of each penalty costs, a different weight on each cost was also determined [18].

2.3 System overview

Figure 4 illustrates the reinforcement learning-based trajectory optimization framework for vehicle mounting. In this framework, initial hip and end-effector trajectories were generated by the path planner module. The main purpose of the planning is to achieve a guideline sequence of hip (close to CoM) and expected contact positions for the given task. Therefore, various planning algorithms can be applied to design the trajectories in the module. Bouyarmane et al. [13] used existing algorithms such as RRTs or PRMs to plan such guide paths and to search multi contact positions in their framework. Qiu et al. [19] reconstructed motion based on real human motion data recorded in motion capture experiments and pre-defined an initial sequence of support configuration.

In this study, initial guideline trajectories are planned from a recorded human movement in mocap. As underscored in Sect. 1, a humanoid with limbs of limited force and torque has similar difficulties with the elderly or the physically impaired. Observation of humans who carefully ingress by constraining limb positions can provide good reference trajectory to those with limited range-of-motion. More details with the capture system and how human movements were recorded are in Sect. 3. With pre-determined frequency, discrete frames were extracted from the initial trajectories and they generated guideline key frame trajectories (both for hip and end effectors).

When it is necessary, the guideline trajectories are post-processed. There are kinematical differences between the human skeleton and humanoid's mechanical body structure. As such, guideline trajectories can not simply mimic the human's raw trajectory. The guideline trajectory should be post-processed to compensate for the differences. Section 3 demonstrates more details of guideline path design using mocap and its post-processing.

After post-processing, both guideline hip and end-effector trajectories searched their neighboring points within limited bounds per time step. Sets of neighboring points were built by combining each searched point from hip and end-effectors at each time step. They became states in the corresponding time step in the action value table (Q value table). All sets in same time step become states in the same column in the action value table. This process was repeated until the last column of the table. The last column means the last time step of the guideline trajectories. In this way, the guideline hip and the end-effector trajectories could be assigned to a reinforcement learning agent with neighboring points of input hip and contact positions as input states.

For studies, a 100 ms time step was used for synchronizing human motion capture. Figure 5 shows the Q value

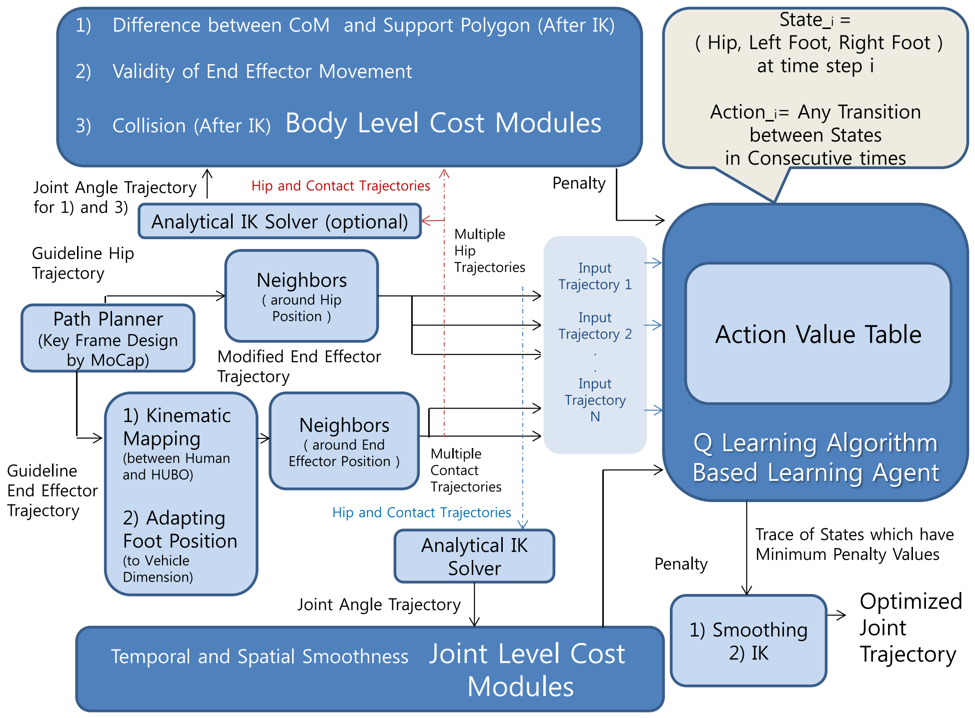
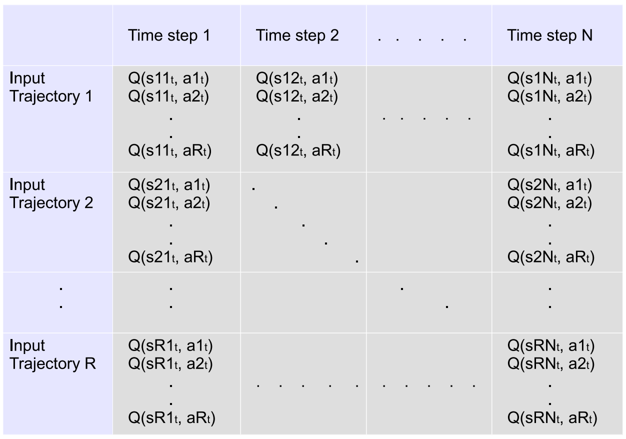

1) Difference between CoM and Support Polygon (After IK)
2) Validity of End Effector Movement
3) Collision (After IK)
Body Level Cost Modules
State_i = (Hip, Left Foot, Right Foot) at time step i
Action_i= Any Transition between States in Consecutive times
Joint Angle Trajectory for 1) and 3)
Hip and Contact Trajectories
Penalty
Analytical IK Solver (optional)
Multiple Hip Trajectories
Guideline Hip Trajectory
Neighbors (around Hip Position)
Input Trajectory 1
Input Trajectory 2
Input Trajectory N
Action Value Table
Path Planner (Key Frame Design by MoCap)
Modified End Effector Trajectory
1) Kinematic Mapping (between Human and HUBO)
2) Adapting Foot Position (to Vehicle Dimension)
Neighbors (around End Effector Position)
Multiple Contact Trajectories
Q Learning Algorithm Based Learning Agent
Guideline End Effector Trajectory
Hip and Contact Trajectories
Analytical IK Solver
Penalty
Trace of States which have Minimum Penalty Values
Joint Angle Trajectory
1) Smoothing 2) IK
Optimized Joint Trajectory
Temporal and Spatial Smoothness
Joint Level Cost Modules

Fig. 4 Trajectory-optimization framework based on Q learning agent



	Time step 1	Time step 2	. . . . .	Time step N
Input Trajectory 1	$Q(s11_t, a1_t)$ $Q(s11_t, a2_t)$ . . $Q(s11_t, aR_t)$	$Q(s12_t, a1_t)$ $Q(s12_t, a2_t)$ . . $Q(s12_t, aR_t)$	. . . . . .	$Q(s1N_t, a1_t)$ $Q(s1N_t, a2_t)$ . . $Q(s1N_t, aR_t)$
Input Trajectory 2	$Q(s21_t, a1_t)$ $Q(s21_t, a2_t)$ . . $Q(s21_t, aR_t)$	. . . .	.	$Q(s2N_t, a1_t)$ $Q(s2N_t, a2_t)$ . . $Q(s2N_t, aR_t)$
. .	. .		. .	. .
Input Trajectory R	$Q(sR1_t, a1_t)$ $Q(sR1_t, a2_t)$ . . $Q(sR1_t, aR_t)$	. . . . .	. . . . . .	$Q(sRN_t, a1_t)$ $Q(sRN_t, a2_t)$ . . $Q(sRN_t, aR_t)$

Fig. 5 Q value table at the learning time t. Every bin in the each column of the table become graded in the same learning time

table at learning time t in this case. Hip and feet position points, $R1$ and $R2$, respectively, were considered from guideline trajectories. This resulted in $N\times$ 100 ms to finish the guideline trajectory and $R1 \times R2(\text{left}) \times R2(\text{right}) \times N$ different states in the Q value table. As such, time duration of planned motions directly affects the dimensionality of state-and-action domains. The selection of neighbor configuration which is important factor defining the exploration potential (of the presented approach) is also highly related to dimensionality. More details with this feature is described in Sect. 6. With analysis of trade-off between computational cost and exploration power of the approach, the optimal sampling frequency and neighboring bounds are determined in the section. Experimental results with those values presented that computation time of the approach can provide compatible performance with other existing approached under the system with similar specifications.

In this study, to decrease the size of the Q value table, just foot movements were planned for contact trajectories of end-effectors and used for the optimization system. Decreasing the table size could reduce computation time for the optimization process. Humanoid fingers lack gripping power and the dexterity needed to grasp and hold onto complex shaped objects in a vehicle [13]. Therefore, by not considering hand-contacts in this analysis, the problem of finger gripping was simplified. However, this does not mean that the presented approach can not plan a optimal sequence of hand movement which has safe contacts and assists ingress task. With necessity, each hand position can be readily collected from guideline trajectories and their neighboring points, $R3$ and $R4$, respectively, can be collected from the trajectories. This additional process results in $R3(\text{left}) \times R4(\text{right})$ multiplier to the existent dimension of Q value table. This issue is further discussed in Sect. 4.3.

Figure 6 illustrates the state-building process from three candidate points (position of the hip and feet). Each point is extracted from their respective neighbors. The hip's state is defined by its position. In contrast, the state of each foot has both position (3-DoFs) and rotation (1-DoF) to account for leg rotation of the yaw axis. These sets of extracted neighboring points became input states for the reinforcement learning algorithm. In other words, one set of guideline hip and end-effector trajectories generated multiple input trajectories for

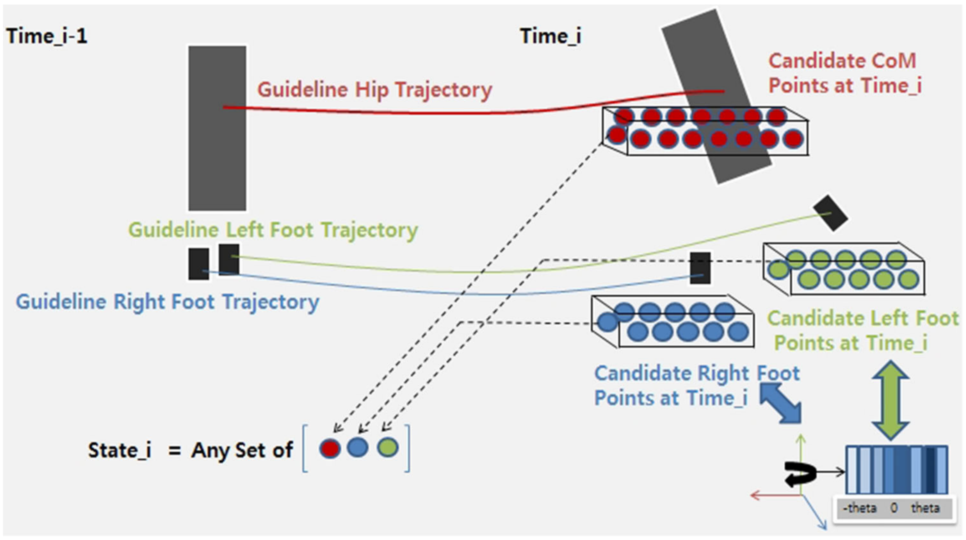

Time_i-1
Time_i
Guideline Hip Trajectory
Candidate CoM Points at Time_i
Guideline Left Foot Trajectory
Guideline Right Foot Trajectory
Candidate Left Foot Points at Time_i
Candidate Right Foot Points at Time_i
State_i = Any Set of
-theta 0 theta

Fig. 6 A state at time step i which consist of three neighboring points

the learning agent. This process made the Q table have states which each has different humanoid configuration.

In the Q table of Fig. 5, state smn_t is a set of hip and contact positions from the time step n of the mth set of neighboring points at learning t. The first point is hip position and others are foot positions of one input trajectory at time step n. The Q value table also shows that each state has its own set of actions, a_t. When the current state is smn_t, a possible action for this state is defined as a transition between state smn_t and any of the states in time step $n + 1$.

For example, the current state is smn at the learning time t. If aj is selected as an action for the state, the next state becomes $sjn + 1$, which is a set of hip and foot positions at time step $n + 1$ of the jth input trajectory.

Using Q value iteration equation, $Q(smn, aj)$ is updated based on its previous value and the minimum predicted future Q value.

At the learning time t, every other pair of state-and-action in the column n of the Q table become also graded by the Q learning algorithm. With the inverse grading rule (described in Sect. 2.2), a set of state-and-action pair in the column $n - 1$ repeats the whole process at the next learning time $t + 1$. After grading bins in the first time step, the learning process is terminated and the generated action value table becomes fully graded.

2.4 Cost function modules

To update a Q value of each bin in the Q table, various penalty cost functions were considered in the optimization framework. The penalty values were largely divided into two different groups. The first group of the penalty values was measured in a robot body level. The cost functions in this category calculated a robot's static balancing ability and internal and external collisions. Validity of the end-effector movement was also checked in this level. More details of the cost functions in the robot body level are given in Sect. 4. The next group of penalty values was measured at the robot joint level. For this, bins in the Q value table passed an inverse kinematics (IK) process and joint angle values were calculated. Various penalties such as joint angle and velocity limits were computed from these angle values. Energy consumption such as torque limit obedience was also checked at the joint level. Section 5 describes more details of the cost function modules at the joint level.

As such, each cost function module assigned a penalty value to a bin of the Q table if the bin has a pair of state and action $((smn, aj))$ which violates the corresponding constraint. Every Q value of the table became updated with those assigned penalty values. Different weights on each penalty value can generate different output trajectories in the learning agent. Therefore, weighting factors on each cost function were determined based on the relative importance of each penalty [18].

2.5 Output trajectory generation and adaptive components

After several iterations of grading all the bins in the Q table, a search for the minimum penalty value for state-and-action pairs was conducted at each time step. The selected pairs became new key frames of the vehicle mounting motion. The key frames were then integrated to generate a smooth motion.

The trajectory optimization framework has several adjustable components. Cost function modules, post processing and the reinforcement learning algorithm are examples. Those components can adaptively fit requirements of a given task while keeping the overall structure of the framework.

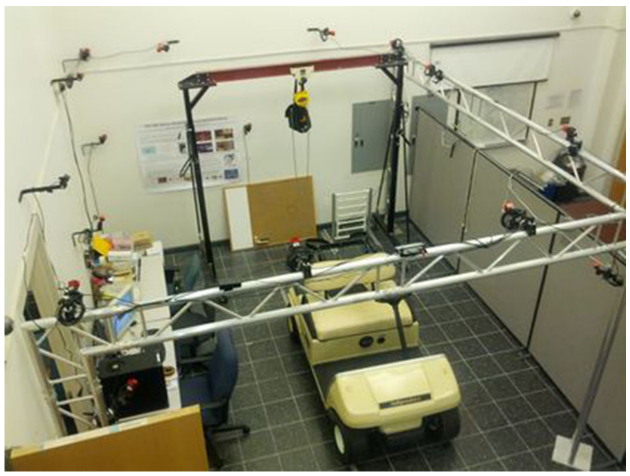


Fig. 7 Optitrack motion capture system

In a previous study [18], the authors presented an approach to generate and optimize a humanoid's object-reaching motions. For penalty values, only joint level cost functions were considered in the learning agent. However, in this study, various cost functions are added for optimizing the vehicle mounting motion; the framework's overall structure is kept and only cost functions in the robot body level are added. Compared to the previous work which just focused on only the upper body's reaching motion, this extended framework satisfies whole-body constraints such as static balancing and collision-free trajectories. For post-processing and reinforcement learning, specific algorithms can be chosen among various candidate techniques based on the task's characteristics. The net effect is that the trajectory optimization framework can adaptively generate an output trajectory that has minimum penalty values for defined constraints.

3 Path planner

3.1 Motion-capture system

Initial hip and end-effector trajectories were generated by the path planner module in the trajectory optimization framework. A ground vehicle is designed for the human body. As such, its structure is often optimized for the human driver's mounting [20]. Therefore, the human's motion can provide a good guideline trajectory for humanoids.

To record the mounting motion, the authors used mocap. Figure 7 shows the mocap which consists of 18 Optitrack FLEX:V100R2 cameras. The system can capture a 12 × 10 square foot area at a maximum sampling frequency of 100 Hz. Figure 8 demonstrates markers which were attached on the human body (left) and the marker's rigid bodies (right).

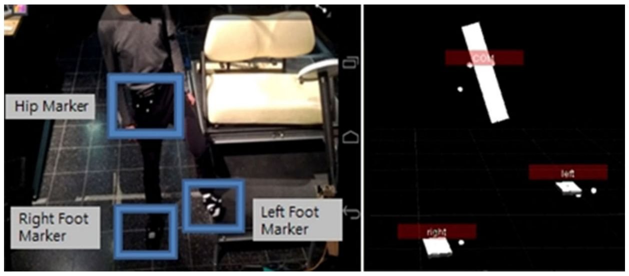


Fig. 8 Markers for recording ingress (*left*) and rigid bodies for hip and each foot (*right*)

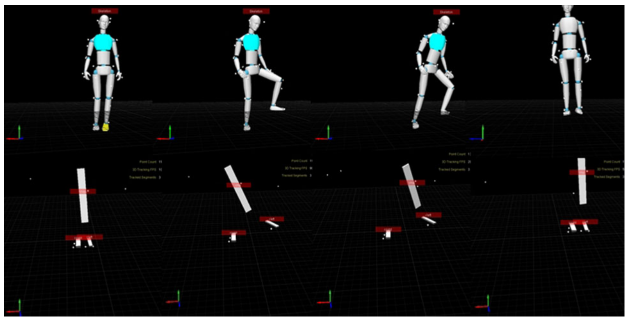


Fig. 9 Recorded human body during ingress (*top*) and rigid bodies of the captured motion (*bottom*)

The movements were captured at a 100 ms sampling time. Figure 9 demonstrates the recorded human body model and its selected rigid bodies (hip and feet) during ingress.

With the pre-determined frequency (10 Hz), discrete frames were extracted from the recorded human movement. As such, guideline key frame trajectories (both for hip and end-effectors) were designed as initial inputs of the framework.

The main purpose of the path planning module is to design a sequence of initial hip and contact positions for a given task. Therefore, other existing sampling algorithms such as RRT also can be used in the presented framework depending on necessities and characteristics of the task (with previously known dimension and kinematics of vehicles). Like other optimization approaches [13,19], the objective of this approach is to compensate kinematic, dynamic and collision constraints of initial guideline paths.

3.2 Post-processing

After the capture, a post-processing stage was implemented for kinematic mapping between the captured rigid bodies and humanoid. A Hubo+ model is used in this study. This model has 38-DoF in the body. Additionally both hands have 1-DoF in each of the five fingers. Figure 10 shows a robot's all links and joints.

The process compensated for kinematic differences between the human body and the robot's mechanical struc-

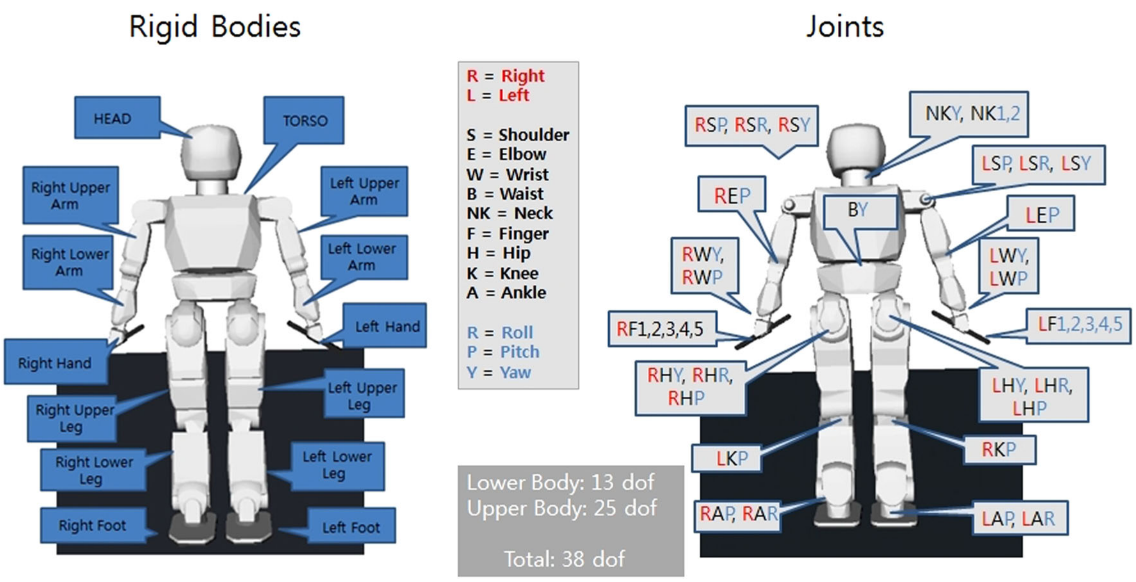


Fig. 10 Links and joints of the robot

ture. To achieve the goal, a calibration stage which a person stands on the ground with an upright posture was conducted in mocap area. A set of relative position values between the rigid bodies was calculated first. The values were then compared with the corresponding kinematic values of the robot. Differences between those two sets were used for computing offset values between the human body and the humanoid's structure. The offsets edited the end-effector's guideline trajectories (foot trajectories in this study).

The vehicle's kinematic dimensions, like roof height and seat position, also redesigned the guideline trajectories. Based on such dimensions, hip and foot position values were modified from the guideline trajectories. This process made the guideline trajectories fit the vehicle's kinematic features more appropriately.

This process increased the probability of finding better hip and foot positions in the given vehicle. In the optimization framework, a collision check between the humanoid and the vehicle is one of the cost value functions. Therefore, post-processed guideline trajectories can provide states which have smaller penalty values in the Q value table.

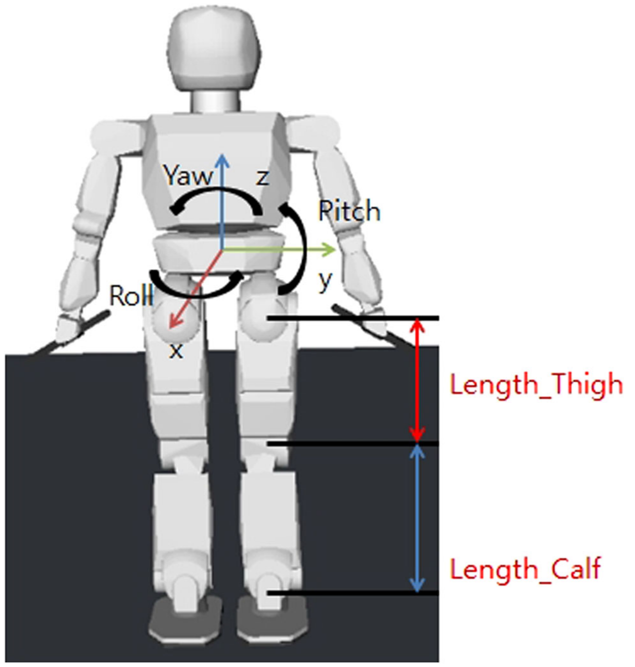


Fig. 11 Robot's body fixed coordinate

4 Penalty functions at the body level

4.1 Static balancing ability

Each bin (a set of a state and an action) in the Q value table was checked whether it can generate a static balancing posture. For this, the robot's CoM position was calculated first.

Each hip and foot point in the state has its own position (3-Dofs) and rotation (1-Dof) values. Joint angles were calculated from the values using an IK process. [21] demonstrates the process in details. Figure 11 shows the robot's body fixed coordinate and two necessary constants (LC = length calf, LT = length thigh) for the analytical IK process.

$$\mathrm{CoM} = \frac{\sum_{i=1}^{n} m_i * r_i}{M} \quad (1)$$

$$(\mathrm{CoM}_x, \mathrm{CoM}_y) \subset (f(x, y)) \quad (2)$$

For each link, its coordinate was calculated based on the joint angles and the supporting foot's position. The supporting foot is the one which is in contact with the ground. After the forward kinematics (FK) process, the robot's CoM position was computed. Equation 1 shows how the CoM position can be computed when a humanoid consists of n rigid bodies (links). Each body has mass m_i and coordinate r_i. M is total mass of the humanoid. As mentioned in Sect. 2.3, only foot movements were planned for the contact trajectories of end-effectors in this study. Therefore, joint angles of the upper body did not change. The net result is that the whole upper body was treated as a single link for the CoM position calculation.

The CoM position was then compared with a support polygon to check whether the state satisfies a static balance criterion. The CoM must lie in the polygon to achieve static stability. Therefore, the bin which has a state that does not meet the balance criterion was assigned a penalty. Equation 2 demonstrates the relationship between the CoM position and the supporting polygon $f(x, y)$. In a single support phase (SSP), $f(x, y)$ is a contact surface of one supporting foot. In a double support phase (DSP), $f(x, y)$ is a convex hull of both feet on the horizontal surface.

4.2 Self and external collision checks

A collision checking process was also integrated in the optimization framework as a cost function module. For each bin in the Q value table, internal and external collisions were checked. Each link's position which was calculated from the FK process (see Sect. 4.1) was used again for this collision computation. For simple collision detection, a bounding volume method is used in this study. When two bounding volumes do not intersect, then the contained objects cannot collide into each other. As can be seen in Fig. 12, bounded spheres were serialized in the robot model and the ground vehicle model (golf cart). To check external collisions, Euclidean distances between two sampled spheres (each from the robot and from the vehicle) was computed for every possible pair. If the distance is bigger than the sum of radii, the pair was assumed to be collision free. Internal (self) collision detection was implemented in the same manner between the bounded spheres of the robot. The bin which has a state that does not meet the collision criterion was assigned a penalty.

4.3 End-effector movement checks

To produce valid states, the planner must ensure that a state change does not violate contact constraints. Any chosen action must not cause translational or rotational movements of the supporting foot. On the physical robot, this constraint is due to contact friction. Any attempted movement of a support foot could cause the foot to break contact prematurely. Therefore, the bin in the Q table which has the supporting foot movement was assigned a penalty. For the moving leg, a Euclidean distance between the state (in the bin) and a future contact position was used as the penalty. This penalty accelerated foot movement of the moving leg and convergence to a tentative goal position. This future goal position was determined from the initial contact trajectory (feet) which was captured from mocap in Sect. 3. Figure 13 demonstrates one example of non-valid foot movement of a supporting leg which is contact with the ground.

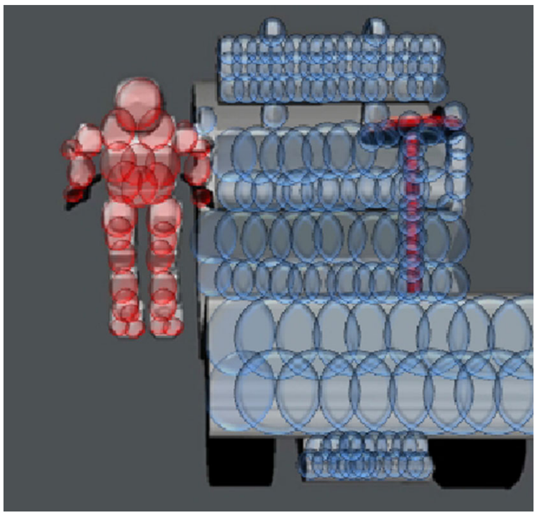


Fig. 12 Bounding spheres in a robot and a golf cart for collision checks

As described in Sect. 2.3, each end-effector bring multiplier to the existent dimension of Q value table. Therefore, including additional feet or hands can increase computational time of this approach. However, this issue can be mitigated by not computing penalty values of state-action pair only for the supporting end-effector. Any action which caused movements of the supporting foot got penalties as shown above. It means that position and rotation of the supporting end-effector do not need to be considered as an valid input state in the Q value table in this approach. Therefore, Q values of bins which have the supporting end-effector (as its input state) do not need to calculate penalty values. In mounted mobilities such as vehicle ingress or ladder climbing, at least two end-effectors are fixed onto their contact spots. Therefore, there are multiple supporting end-effectors per time step during motion. By not computing their penalty values, actual size of Q value table can be determined with only hip and flying end-effectors per time step. It results in reduction of computation time for learning process.

5 Penalty functions at the joint level

5.1 Energy consumption and torque limit

To predict energy consumption of each bin in the Q value table, it was necessary to calculate torque values of every joint

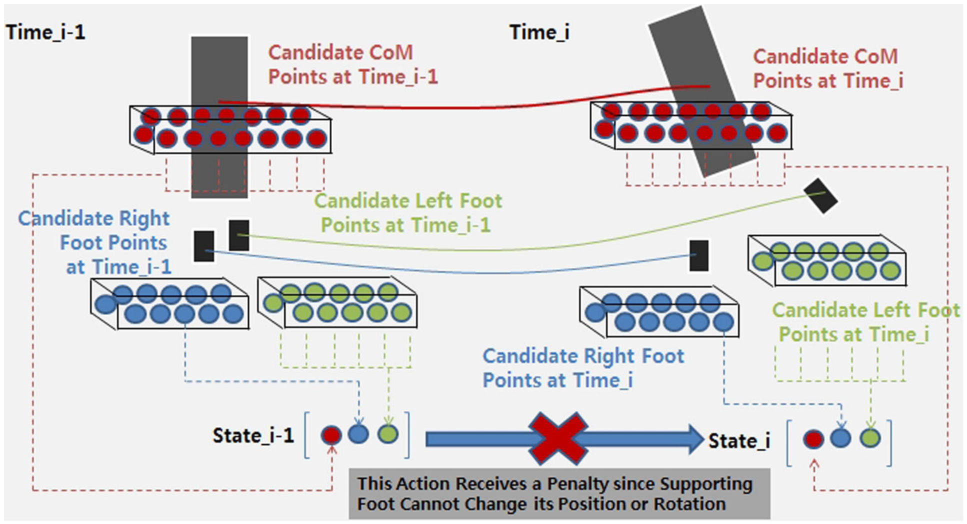

Time_i-1
Time_i
Candidate CoM Points at Time_i-1
Candidate CoM Points at Time_i
Candidate Left Foot Points at Time_i-1
Candidate Right Foot Points at Time_i-1
Candidate Left Foot Points at Time_i
Candidate Right Foot Points at Time_i
State_i-1
State_i
This Action Receives a Penalty since Supporting Foot Cannot Change its Position or Rotation

Fig. 13 Validity of end-effector movements check

in a robot. To relate the joint torques and joint angles, ProPac [22], a Mathematica package, was used. ProPac supports the assembly of various simulation models for mechanical systems such as ground vehicles, aircraft, or robots. In this study, it generated the full nonlinear model equations in explicit form for modeling the robot.

To build the robot model, all necessary data for individual joints and links were collected using Open Inventor, a CAD toolkit. Mass, CoM or moment of inertia are examples of the data collected. Then, a system interconnection structure was created from a joint hierarchy of the robot. Figure 10 shows all defined links and joints for the robot. The ProPac model needs one fixed ground body (reference frame) for a modeling process. Therefore, a separate model which has the left or right foot as the grounded body was made. The model was then selected depending on the supporting leg of the bin.

With the built model, ProPac calculated all components of the Poincare' equation [22]. The recursive multi body dynamics equation for serial open and branched kinematic chains was formulated using Lie group and Lie algebra. First, the kinematics of an open chain was modeled as a sequence of homogeneous transformation between consecutive joint frames. Spatial velocity of the branched chains were then calculated. Last, the inward recursion of forces and torques was calculated with the chosen external forces. Suleiman et al. [11,12] provide more details with how the Poincare' equation can be formulated to relate the joint torques and joint angles explicitly for humanoids structure.

Equation 3 shows the generated dynamic equations for the robot model. q is the generalized coordinate vector of a joint angle and p is a quasi velocity of a joint.

$$M(q)\dot{p} + C(q, p)p + F(q) = B_p \tag{3}$$

where

$$C(q, p) = -\left[\frac{\partial M(q)p}{\partial q} V(q)\right] + \frac{1}{2}\left[\frac{\partial M(q)p}{\partial q} V(q)\right]^{\mathrm{T}} + \left[\sum\nolimits_{j=1}^{m} p_j {X_j}^T\right] V^{-\mathrm{T}}$$

$$F(q) = V^{\mathrm{T}}(q)\frac{\partial u(q)}{\partial q^{\mathrm{T}}}, \quad B_p = V^{\mathrm{T}}(q)B$$

$u(q)$ is the potential energy function and M is a spatial inertia matrix. B_p denotes the generalized forces represented in the p-coordinate frames and B denotes the generalized forces in the velocity of q coordinate frame. In combination with Eq. 4 which is the kinematic equations of each joint and link, these equations provide a closed set of equations.

$$\dot{q} = V(q)p \tag{4}$$

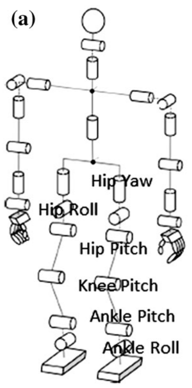
In Q table, each bin has all known values for the joint angle, velocity and acceleration. Therefore, B_p which is a set of generalized motor forces can be calculated for all joints in the robot. The net result is that torque for each joint can be calculated using this inverse dynamics method under a given trajectory with the forward kinematic model. Every notation in the Eqs. 3 and 4 are from [22].

After getting all joint torque values from the selected bins in the Q table, they were compared with torque limits. In this study, a continuous torque limit of the harmonic driver was used for each joint's torque limit. Then, the bin which has an action that exceeds the limits got a penalty. This prevented the generated motion from requiring too much torque or current.

5.2 Temporal and spatial smoothness

Any bins in the Q value table which have an action that exceeds smoothness limits also received a penalty. A joint

(a)



(b)

Joint		Angle Range (Degree)
Hip	Yaw	-110 to 110
	Roll	-30 to 30
	Pitch	-107 to 92
Knee	Pitch	-4 to 149
Ankle	Pitch	-94 to 97
	Roll	-90 to 90

Fig. 14 Robot's joint hierarchy and limits

Joint	Reduction Gear Cont. (Nm)	Reduction Gear Stall (Nm)	Motor Cont. (Nm)	Motor Stall (Nm)
Hip Yaw	27	71	16.25	428.75
Hip Roll	34	95	32.87	867.24
Hip Pitch	34	95	26.00	686.00
Knee	34	95	19.20	548.81
Ankle Roll	27	71	20.54	542.02
Ankle Pitch	27	71	16.25	428.75

Fig. 15 Robot's joint torque limits

angle, velocity and torque's temporal changes are examples of such limits. The weighted sum of these penalties ensured that the final output motion met the physical and dynamical limits of the robot. Penalizing actions which exceed the joint angle limit ensured a smooth trajectory over time. Limiting the rate of torque change ensured spatial smoothness. Figures 14 and 15 illustrate the robot's kinematic and torque limits of each joint. In this study, a 100 ms time step was used for getting key frames from an initial path-planned trajectory. For joint limitations which are defined under fixed time duration (10 ms), rescaling was implemented based on the sampling time of key frames. Joint velocity, acceleration and the rate of torque changes are the examples. Those values were multiplied with 10 to integrate the time gap before use in the cost function module.

6 Output trajectory generation and computational complexity

6.1 Output trajectory generation process and analytic verification

After grading all the bins in the Q table, a search for bins with minimum penalty values was executed at each time step (see Fig. 2). The selected bins were treated as new optimized key frames. They were integrated to generate the smooth trajectory. For this, the velocity and acceleration limits of each joint were used again. Lastly, after the IK process, these integrated key frames become the joint angle trajectory.

To test a validity of the proposed trajectory optimization framework, the authors built an experimental scenario which follows this sequence: (1) a robot lowers its pelvis with vertical direction (Z axis) initially; (2) the robot then moves its pelvis to the right side (Y axis) by swaying hip. The executed motions are demonstrated in Fig. 16.

To design initial mocap trajectories for motions above, the authors used mocap. The movements were captured at a 100 ms sampling time. Figure 17 demonstrates the CoM (Y axis) and pelvis (Z axis) position of the recorded human movement. The characteristics of the hip movement are: -125 mm in the Y axis and -150 mm in the Z axis. Figure 18 shows the joint angle trajectory which is calculated from the mocap motion.

Mocap is a direct approach to implement motion planning in complex spaces. However, a kinematic difference between the human body and the mechanical structure of humanoids is a significant limitation. The motion-captured trajectory does not guarantee static balancing of a robot and

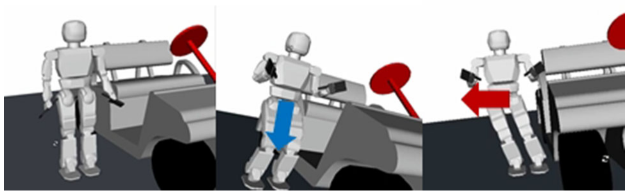


Fig. 16 Swaying movement of a robot

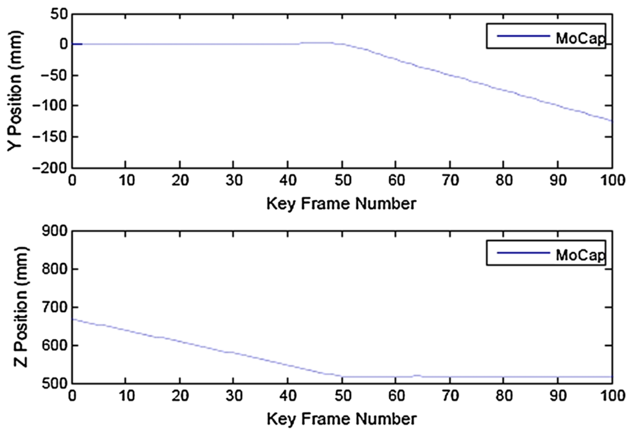


Fig. 17 CoM and Pelvis trajectory from recorded human movement

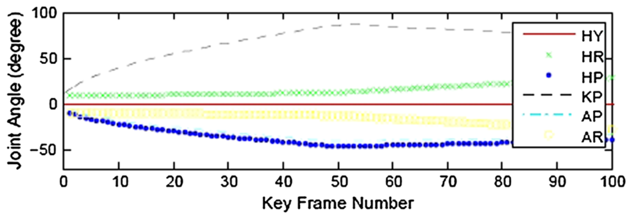


Fig. 18 Joint angle trajectory for right leg (*HY* hip yaw, *HP* hip pitch, *HR* hip roll, *KP* knee pitch, *AP* ankle pitch, *AR* ankle roll)

also obeys angle limits of some joints. To overcome such limits, captured motions should be scaled with desired kinematic and dynamic constraints.

When the selected constraints (joint and static balance limits in this analysis) are given like Eq. 5, where

$$\left.\begin{aligned} q_{t0} = q_0, \dot{q}_{t0} = 0, \ddot{q}_{t0} = 0 \\ q_{tf} = q_0, \dot{q}_{tf} = 0, \ddot{q}_{tf} = 0 \\ q^- <= q_t <= q^+ \\ \dot{q}^- <= \dot{q}_t <= \dot{q}^+ \\ y^- <= \mathrm{CoM}(q_t)_y <= y^+ \end{aligned}\right\} \quad \text{Constraints} \qquad (5)$$

the performance error of the recorded human movement is measured by the R^2 Norm method [23]. The error was then averaged by sampling time (100 ms in this study). $q_t$is the joint positions of the robot. q^-, q^+, $\dot{q}^-$ and $\dot{q}^+$ are chosen according to Figs. 14 and 15. Bounds for $CoM(q_t)_y$ is determined based on the position of supporting foot and the foot's width.

The average distance error between the recorded CoM position and the balancing limits is calculated as 9.2321 mm. The averaged joint angle error between the recorded movement and the joint limits is calculated as 5.4362°. When 0.5 mm and 0.1° are used for threshold values, the mocap trajectory is not acceptable for the use in Hubo+. Peak values of the hip pitch and ankle roll angle are below their minimum limits (−40° and −20° respectively).

The input mocap trajectory is then processed with the reinforcement learning agent-based trajectory optimization framework. Figure 19 presents the results which are generated from the framework. The averaged distance error between the optimized pelvis position and the pelvis limits is calculated as 0.261 mm. The averaged joint angle error between the calculated movement and the joint limits is calculated as 0.097°.

The pelvis and the joint angle errors become decreased in the generated trajectories. Both errors have below-threshold costs. Peak values of the hip pitch and ankle roll angle also meet their minimum limits. It resulted in less amounts of the pelvis movement as can be shown in Z axis trajectory of Fig. 19. The net result is that the proposed method can optimize the input raw trajectory under a given set of various constraints.

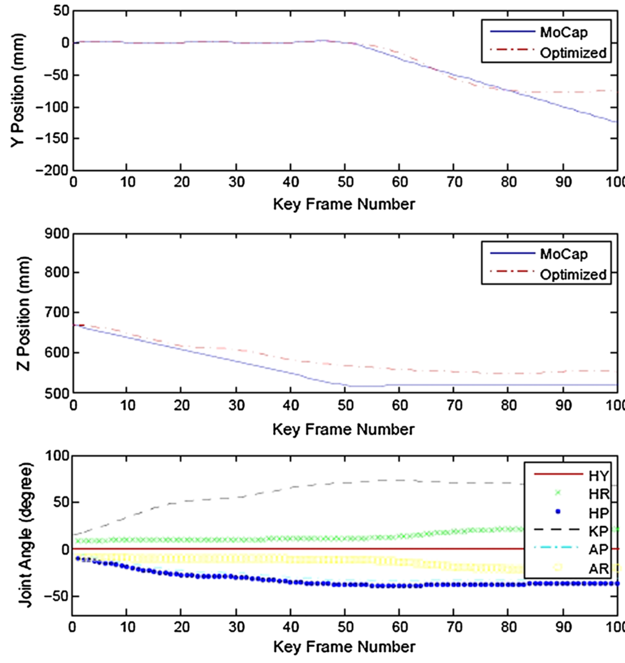


Fig. 19 Trajectories after reinforcement-learning optimization

6.2 Computational complexity and processing time

In this study, the initial guideline trajectory was modified during the post-processing stage for kinematic mapping. The trajectory was then optimized with respect to all defined constraints in the optimization stage. For this, a system which has specifications of 1.8 GHz i-5 processors, 6 GB of memory and Linux OS was used.

As described in Sect. 2.3, computation speed of the optimization process was determined based on the number of states which were to be optimized. Therefore, total time duration in the guideline trajectory affected computation time. Neighboring bounds also made an effect in computation time of the optimization process.

Though time duration of planned motions increase the dimensionality of state-and-action domains, this could be mitigated with sampling process. Figure 20 shows estimated processing time based on various time durations and

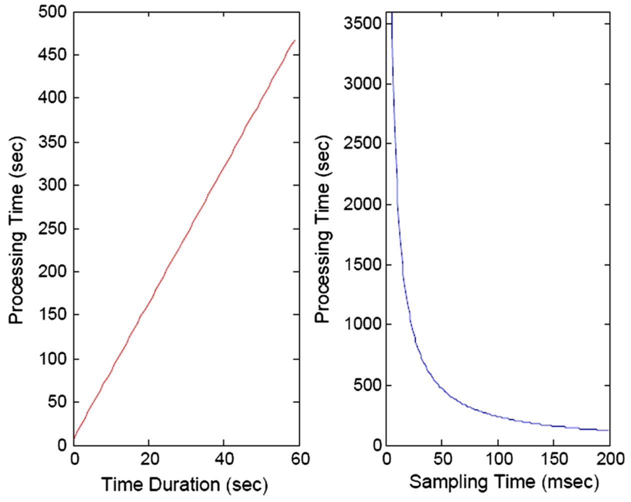


Fig. 20 Effects of time duration (*left*) and sampling time (*right*) on processing time

sampling frequencies of planned motions. The right graph of Fig. 20 demonstrates that the processing time become decreased with higher sampling time of the guideline trajectory. For this study, a 100 ms time step was used for getting initial key frames from the guideline trajectory. The value was chosen based on the maximum value which does not increase performance errors which account for multiple time-varying constrains of the robot such as joint limits and collision avoidance. As described in Sect. 5.2, time duration-related joint constraints were rescaled according to the sampling frequency of initial key frames. Section 6 demonstrated that the finally selected bins (after grading) were integrated to generate the smooth trajectory considering the velocity and acceleration limits of each joint. Therefore, temporal and spatial smoothness of each joint are kept regardless of the initial sampling time. However, performance errors from collision constraints were dependent on the sampling frequency of initial key frames. Therefore, the frequency was tuned to find the maximum value which does not increase the performance error.

The selection of neighbor configuration is also highly related to dimensionality. It is an important factor defining the exploration potential of the presented approach. Figure 21 shows the estimated processing time and performance error (in static balancing limits) based on various neighboring bounds. With resolutions of 10 mm in transition and 1° in rotation axis, 800 neighboring samples were optimized per time step from the guideline trajectory in this study. With the input ingress trajectory which has 30–32 s execution time (with 10 Hz sampling frequency), whole optimization process took about 4 min. The left graph of Fig. 21 demonstrates the processing time linearly increases with the number of neighbors. However, the right graph shows that the performance error become converged below its threshold (0.5 mm in this study) after 800.

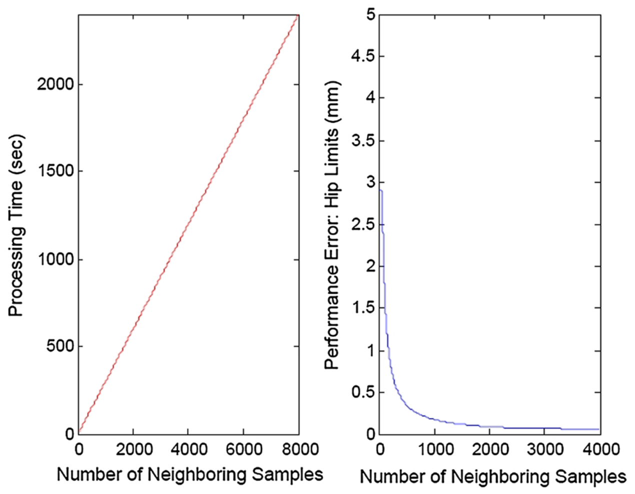


Fig. 21 Effects of neighboring bounds on processing time (*left*) and performance error (*right*)

Finally, the whole processing time for both post-processing and optimization was ranged between 210 and 250 s. 800 neighboring samples were optimized in each time step from the guideline trajectory which has 30–32 s execution time (with 10 Hz sampling frequency). The net result is that it took about 4 min for the robot to have the optimized vehicle mounting trajectory from the guideline motion. In similar studies, [19] found paths for the ingress motion in about 7 min (Xeon 3.4 GHz Processor and 8 GB of RAM) and [13] took about 20 min for the computation time of optimizing locomotive motions (Intel 1.70 GHz i5-2557M Processor). Considering the fact that initial guideline trajectory planning process took about 3 min, the approaches took about 4–17 min for the optimization process. It presents that computation time of the learning agent-based approach can provide compatible performance with other existing approaches under the system with similar specifications.

7 Experimentation result

There are two phases of ingress, which can be thought of states. The robot should progress through each phase towards task completion. First phase is *Step*: step up on a vehicle floor, lower completely onto seats and move to the optimal driving position. Second phase is *Interface*: put one foot on a brake pedal and grasp a steering wheel with one or two hands. In case of egress, phase *Step* can be replaced with *Step down*: go from sitting to upright position and get feet from vehicle floor down to ground.

In this study, the authors focused on planning and optimization of motions for *Step* phase. The *Step* phase consists of different parts which include *Step up* and *Sit down and Scoot* motions. During the record in the capture system, the human was asked to step on the floor of the given vehicle, to lower onto seats and to scoot itself to the driver's seat. The captured motions become guideline trajectories for the reinforcement learning-based trajectory optimization framework.

The optimized trajectory was then tested using a virtual robot model in a simulation environment, which is called, OpenHubo. OpenHubo is an Open Robotics Automation Virtual Environment (RAVE) [24] based simulation tool which is developed by Drexel University. It provides motion planning and control of the robot model in a virtual environment [25] featuring kinematic display, physics-based simulation and simulated sensors. OpenHubo used Open Dynamics Engine (ODE) for its physic and collision handling.

OpenHubo includes collision meshes, mass and inertia properties for Hubo+ robots. The robot model was initially built by deriving rigid body masses and inertia from a detailed CAD model in SolidWorks. The model was then exported to Universal Robot Description Format (URDF) via the URDF

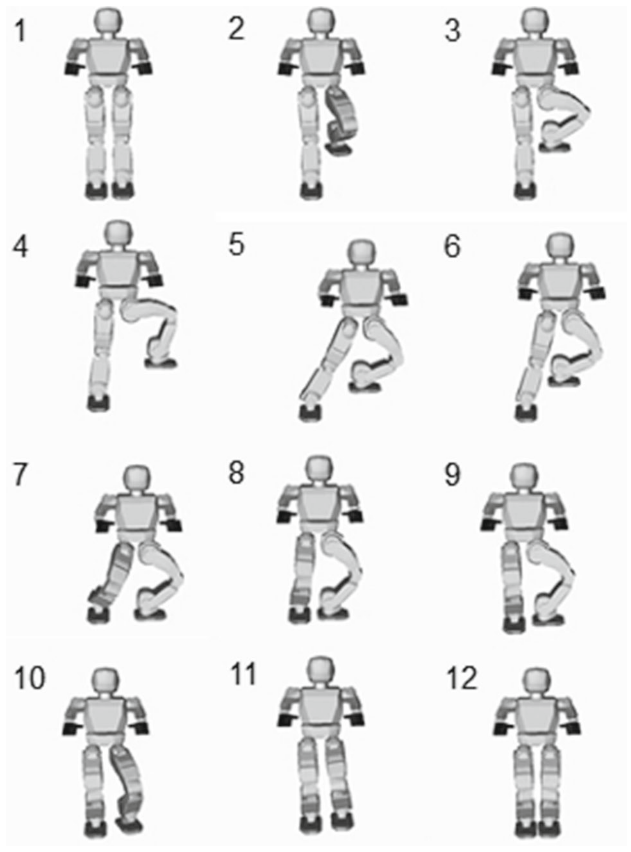


Fig. 22 Guideline MoCap trajectory for *Step up* motion

exporter plugin (in Solid Works). The URDF model was then converted to OpenRAVE XML format using a python module (in OpenHubo) for use in OpenHubo environment [26].

For the vehicle to be mounted, Club Car DS IQ (2008) is used. The electric powered golf cart has a size of 87.6 cm × 97.8 cm. To model the vehicle in OpenHubo, the KinFu module in the Point Clouds Library (PCL) is used [27]. The module stitched multiple views from our depth cameras together and created a high-resolution surface model of the test vehicle. KinFu is an open source version of the original Kinect Fusion algorithm [28].

7.1 Guideline MoCap trajectory

Figure 22 demonstrates the robot's kinematic movement with the guideline mocap trajectory (before the optimization process) for *Step up* motion. It is generated for an initial part of *Step* phase during ingress.

The CoM position often drifts outside the support polygon as shown in Fig. 23.

Also, in many cases, the robot does not maintain its joint limits. Such postures can result in high torque and unnatural movement (Fig. 24).

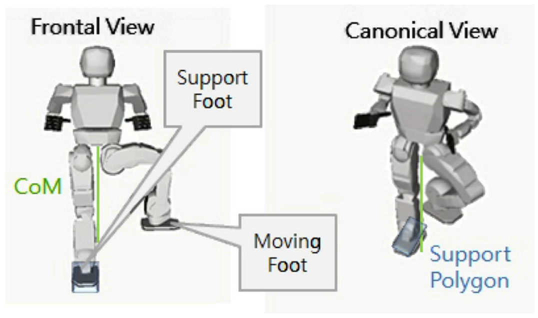


Fig. 23 Comparison between the CoM position and the support polygon

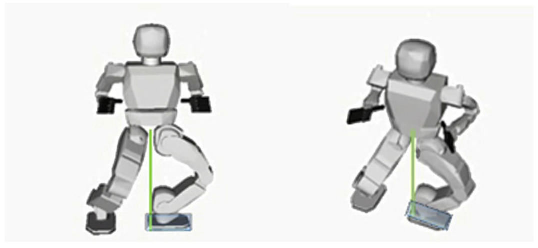


Fig. 24 Joint angle limit obedience due to non natural posture

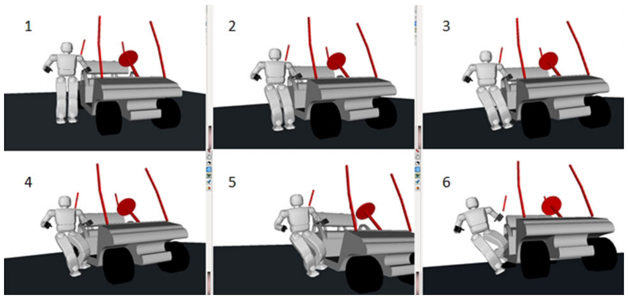


Fig. 25 Simulation result with raw trajectory

Figure 25 shows the physics based simulation result when the guideline trajectory is tested in OpenHubo. The trajectory could not guarantee static balancing and collision avoidance. It resulted in fall of the robot in beginning phase of ingress.

7.2 The optimized trajectory

The optimized ingress trajectories were then tested using a robot model in OpenHubo with physics considered.

Figure 26 shows the robot's *Step up* motion during ingress with the optimized trajectory (from the learning agent).

Figure 27 shows the robot's *Step up* motion from a different camera view.

Unlike the guideline trajectory (see Fig. 23), the robot's CoM position lies in the support polygon with the optimized trajectory (Fig. 28).

The robot often obeyed the joint limits with the guideline mocap trajectory. In Fig. 24, the robot rotated its hip roll

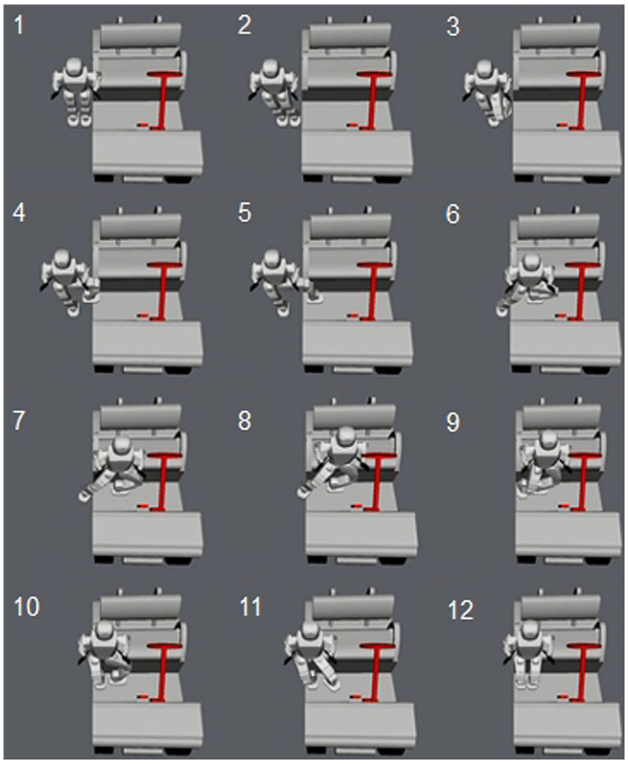


Fig. 26 The optimized ingress trajectory for *Step up* motion

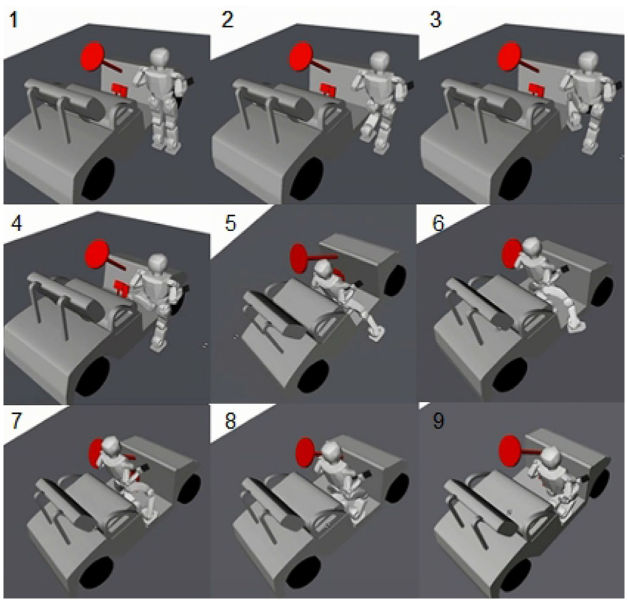


Fig. 27 The optimized ingress trajectory for *Step up* motion from canonical view point

joint more than the limit. It resulted in an unnatural posture which requires high torque and current. However, with the optimized trajectory, the robot changed the hip position and the foot rotation to meet the joint limits. In Fig. 29, the robot turned the right foot to meet the hip joint limits while keeping its balance.

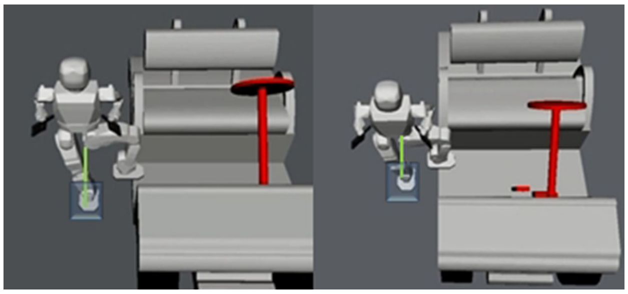


Fig. 28 Comparison between the CoM position and the support polygon

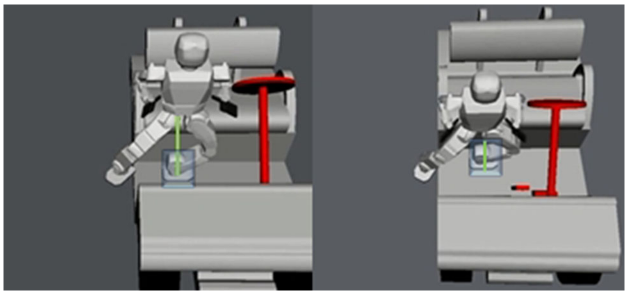


Fig. 29 Heading direction (yaw) change of *left* foot in optimized motion

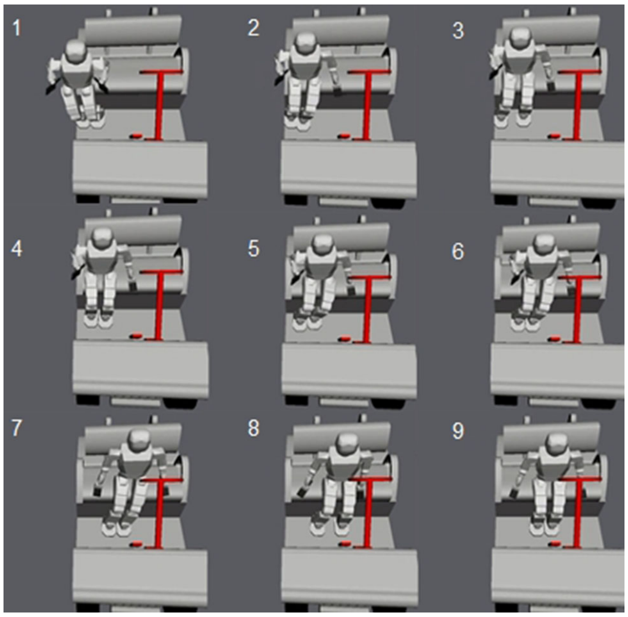


Fig. 30 Optimized trajectory for *Sit down and Scoot* motions

Using the optimization framework and mocap, other vehicle mounting motions were also generated. Figure 30 shows a robot sitting in the passenger-side of the golf cart (*Sit down*) and moving itself to the driver-side (*Scoot*). As a result, the robot completed the entire *Step* phase of ingress. Figure 31 demonstrates the same motion from the canonical view point.

Figure 32 demonstrates the robot's *Step down* motion during egress after driving and arrival to the goal.

Experimental testing and evaluation confirmed the efficacy of the optimized trajectory approach. Figure 33 demon-

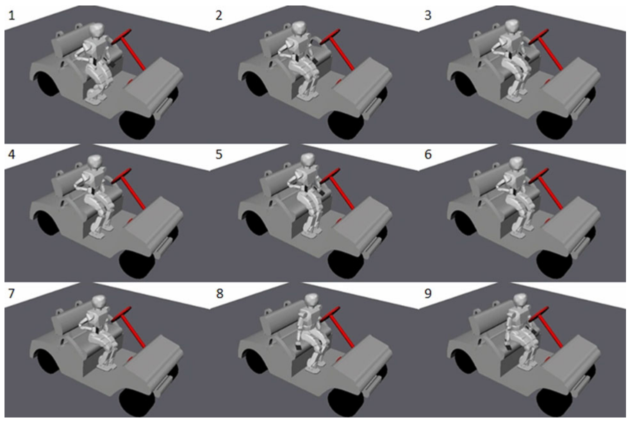


Fig. 31 Optimized trajectory for *Sit down and Scoot* motions from canonical view point

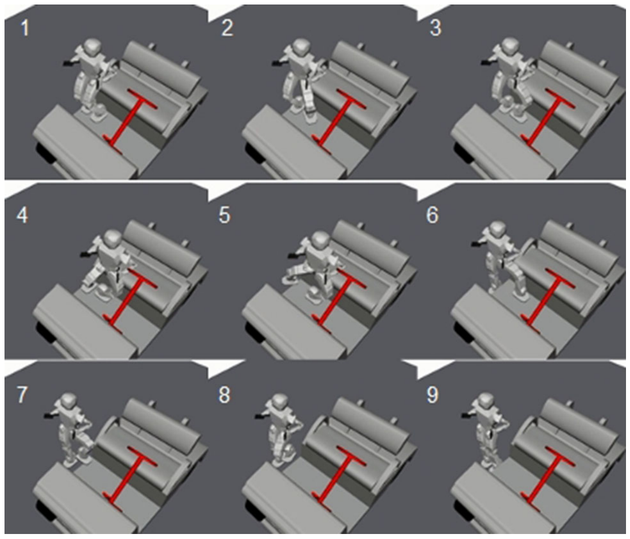


Fig. 32 Optimized trajectory for *Step down* motion during egress

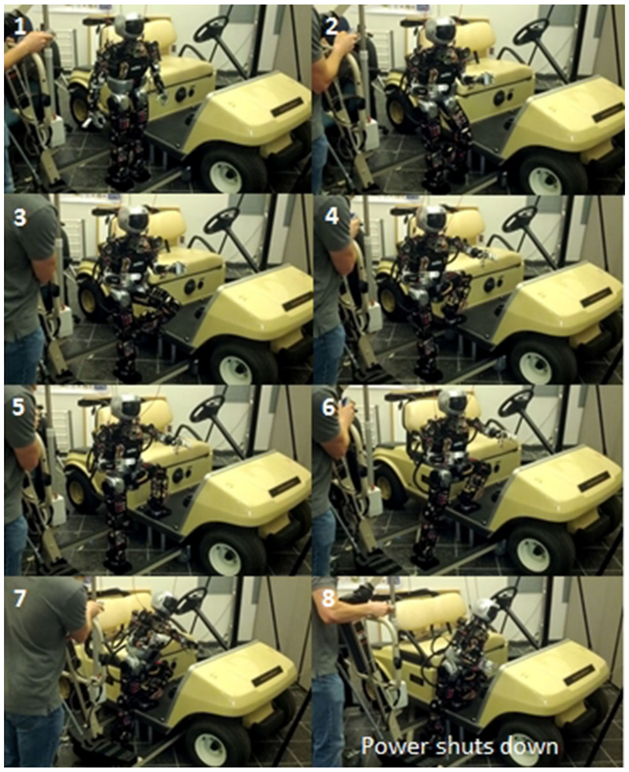


Fig. 33 Verification stage of optimized trajectory for *Step up* motion during ingress [3]

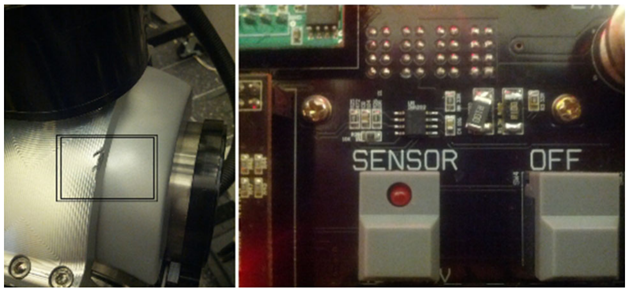


Fig. 34 Practical issues from testing-and-evaluation process: self collision (*left*) and over-heat (*right*)

strates the robot's *Step up* motion during its first half phase. Experiments raised a few challenges like self collisions and over-heat issues. As can be seen in Fig. 33's sub-figure 8, the robot's main power became shut down when its one leg supported whole mass of the robot. Figure 34 shows the addressed problems with more details.

Due to the heavy weights of each leg, a wrench effect was applied to the movement of the robot. It worsen the rigidity of each link pose and caused the unexpected self collisions in some joints such as hip and ankle. Adjustment of the joint angle limit can prevent the connected link from colliding with other parts. However, to climb the high vehicle floor, further decrease of the joint is not possible considering the short leg length of the robot.

High torques in some joints also generated over-heat issues in the power management board of the robot. The applied torques in each joint met the given threshold of the used hardware like harmonic drivers and BLDC motors. However, high currents in the power control board accumulated heats. It generated burnouts of the board. It especially happened during the last half phase of the *Step up* motion which requires the robot to have the knee-bent pose for long time. This unnatural pose is also mainly resulted by the short leg length of the robot.

To solve these problems, technical design requirements which include increases of joint angle limits, torque limits and leg lengths are addressed for the new robot model, called DRC-Hubo. The author's current studies focus on addressing these challenges and further demonstrate the effectiveness of this paper's framework for vehicle handling with DRC-Hubo model [29]. In preliminary results, the robot successfully ingresses and egresses with two different vehicles (golf cart

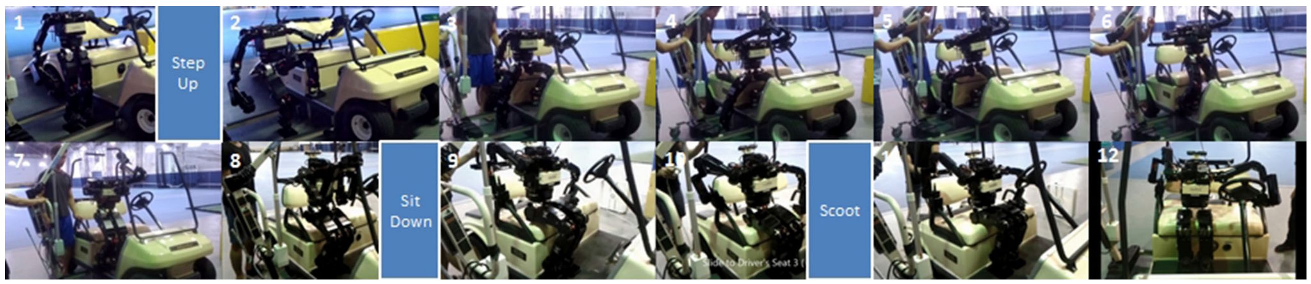


Fig. 35 Vehicle mounting of DRC-Hubo

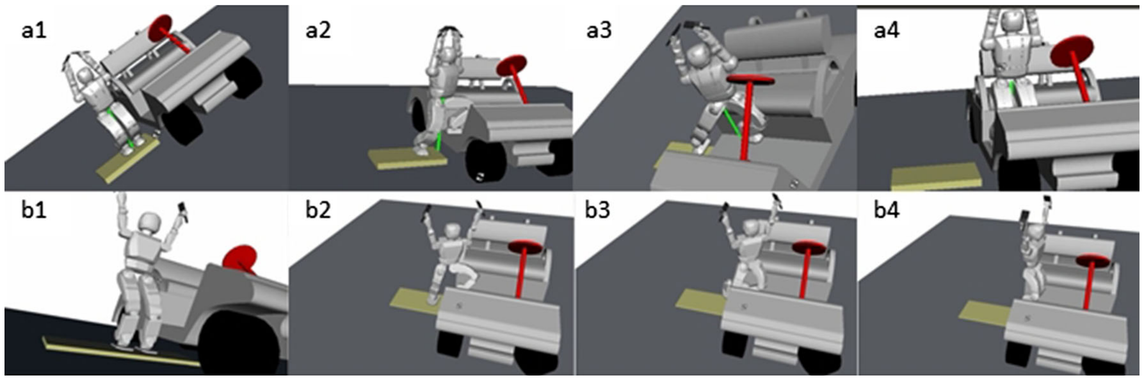


Fig. 36 *Step up* from two different starting positions

and utility ground vehicle) via the planned trajectories from the learning agent-based optimization framework [30]. Figure 35 demonstrates that DRC-Hubo finished *Step up*, *Sit Down* and *Scoot* motions with the optimized trajectory. Followed-up paper will describe how penalty function components of the presented framework are modified to reflect new kinematic and dynamic features of the robot. Furthermore, various vehicle mounting motions (ingress and egress) will be designed and tested through different types of vehicles to prove adaptivity of the presented approach.

Another critical part which should be further study is to generalize the guideline trajectories. As described in Sect. 3, input trajectories are designed by the path planner module in the trajectory optimization framework. They can be generated by motion capture or kinematic path planning algorithms. Since both methods take significant capturing or computation time, it is desirable to re-use initial guideline trajectories which were once designed for a given task. They can be applied to various situations which each has different task conditions such as kinematic change in vehicles or task starting position. For this, path planner module should extract key motion primitives from those initial trajectory and should be adjusted to changes in a given task. In Sect. 3.2, the authors presented that post-processing stage can generate the optimized ingress motions for such different task conditions. Figure 36 demonstrates that Hubo+ can step on the vehicle floor from two initial positions which have different height values. Using a guideline trajectory (which was once designed), foot movements of the trajectory were modified to the kinematic changes and they become optimized through the presented framework. Future study will address this approach further and will extend use of motion primitives for various task conditions in a given task. The net result is that initial path planning can be replaced with simple kinematic processing of initial trajectories.

There are also works to be done for a better robustness of the planning and optimization process in future studies. Currently, a static balancing check based on CoM position is used in the presented framework. For more dynamic motions, other criterion factors such as zero moment point (ZMP) also should be considered. Collision checking and contact-force calculation are other works which also need to be advanced for better performance of this optimization framework.

8 Conclusion

The paper began by underscoring the intellectual merits and broader impacts of vehicle handling (mounting and driving) by a humanoid. Towards this goal, this paper presented a framework to plan and optimize humanoid motions under a variety of internal and external constraints. With the guideline mocap trajectory, the reinforcement learning agent generated the output trajectory which minimizes the penalty values. At the body level, the CoM position was computed for the static balance check. For this, the Hubo+'s various kinematic and dynamic features were used for building a simplified mathematical model. Internal and external collisions and validity of the end-effector's movement were also checked. At the joint level, ProPac was used to build a torque model of the robot to measure energy costs of each joint. Weighting factors on each cost function were determined based on the relative importance of each penalty. The converged Q value table generated the optimized trajectory that satisfies all the defined constraints at the body and joint level. The net effect is a flexible framework towards humanoid vehicle handling. Experiments with the full-sized humanoid verified the approach. The framework has potential for extensions and applications to other humanoids and vehicles. Current efforts include the DRC-Hubo handling a Polaris utility vehicle. The paper's approach was used for DRC-Hubo in DRC-Trials 2013 [31] and will be continuously applied to the 2015 DARPA Robotics Challenge Finals.

References

1. DARPA (2012) http://archive.darpa.mil/grandchallenge/. Accessed Oct 2012
2. Erico G (2013) How Google's self-driving car works. IEEE Spectrum 18
3. NHK Documentary (2013) Future Robot from (TV NHK), 17 March 2013
4. LaValle S, Kuffner J (2000) Rapidly-exploring random trees: progress and prospects. In: New directions, algorithmic and computational robotics, pp 293–308
5. Berenson D, Srinivasa SS, Ferguson D, Collet A, Kuffner J (2009) Manipulation planning with workspace goal regions. In: IEEE international conference on robotics and automation (ICRA), Kobe, Japan, May, pp 618–624
6. Berenson D, Srinivasa SS, Ferguson D, Kuffner J (2009) Manipulation planning on constraint manifolds. In: IEEE international conference on robotics and automation (ICRA), Kobe, Japan, May, pp 625–632
7. Berenson D, Chestnutt J, Srinivasa SS, Kuffner J, Kagami S (2009) Pose-constrained whole-body planning using task space region chains. In: IEEE-RAS 9th international conference on humanoid robots (humanoids), December, pp 181–187
8. Zordan VB, Hodgins JK (1999) Tracking and modifying upper-body human motion data with dynamic simulation. In: Computer animation and simulation 99, Eurographics, pp 13–22
9. Riley M, Ude A, Atkeson C (2000) Methods for motion generation and interaction with a humanoid robot: case studies of dancing and catching. In: AAAI and CMU workshop on interactive robotics and entertainment, Pittsburgh, Pennsylvania, USA, April, pp 35–42
10. Matsui D, Minato T, MacDorman KF, Ishiguro H (2005) Generating natural motion in an Android by mapping human motion. In: IEEE international conference on intelligent robots and systems (IROS), Alberta, Canada, August, pp 3301–3308
11. Suleiman W, Yoshida E, Laumond JP, Monin A (2007) On humanoid motion optimization. In: IEEE-RAS 7th international conference on humanoid robots (humanoids), Pittsburgh, PA, USA, December, pp 180–187
12. Suleiman W, Yoshida E, Kanehiro F, Laumond JP, Monin A (2008) On human motion imitation by humanoid robot. In: IEEE international conference on robotics and automation (ICRA), Pasadena, CA, USA, May, pp 2697–2704
13. Bouyarmane K, Vaillant J, Keith F, Kheddar A (2012) Exploring humanoid robots locomotion capabilities in virtual disaster response scenarios. In: 12th IEEE-RAS international conference on humanoid robot (Humanoids). Osaka, Japan, December, pp 337–342
14. Sutton RS, Barto AG (1998) Reinforcement learning: an Introduction, vol 1, No 1. MIT Press, Cambridge
15. van Hasselt Hado (2012) Reinforcement learning in continuous state and action spaces. Reinf Learn: State Art Springer 12:207–251
16. Watkins CJCH (1989) Learning from delayed rewards. Ph.D. thesis, Cambridge University
17. Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292
18. Sohn K, Oh P (2012) Applying human motion capture to design energy-efficient trajectories for miniature humanoids. In: IEEE/RSJ international conference on intelligent robots and systems (IROS), Vilamoura, Algarve, Portugal, October, pp 3425–3431
19. Qiu Z, Escande A, Micaelli A, Robert T (2012) A hierarchical framework for realizing dynamically-stable motions of humanoid robot in obstacle-cluttered environments. In: IEEE-RAS international conference on humanoid robots (humanoids), Osaka, Japan, 29 November–1 December, pp 867–874
20. Lempereur M, Pudlo P, Gorce P, Lepoutre FX (200) Optimization approach for the simulation of car accessibility movement. In: IEEE international conference on systems, man and cybernetics, Washington, USA, vol 1, October, pp 843–848
21. Jun YB, Oh P (2011) A 3-tier infrastructure: virtual-, mini-, online-hubo stair climbing as a case study. In: Proceedings of biomechanics and robotics, vol 752. ACTA Press
22. Kwatny HG, Blankenship G (2000) Nonlinear control and analytical mechanics: a computational approach (control engineering), 1st edn. Birkhauser, Boston
23. Deza E, Deza MM (2009) Encyclopedia of distances. Springer, Berlin, Heidelberg
24. Diankov R (2010) Automated construction of robotics manipulation programs. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, August
25. Zhang Y, Luo J, Hauser K, Ellenberg R, Oh P, Park HA, Paldhe M, Lee CSG (2013) Motion planning of ladder climbing for humanoid robots. In: IEEE international conference on technologies for practical robot applications (TePRA), Woburn, MA, April, pp 1–6
26. Ellenberg R, Oh P (2014) Contact wrench space stability estimation for humanoid robots. In: IEEE international conference on technologies for practical robot applications (TePRA), Woburn, MA, April, pp 1–6
27. Rusu R, Cousins S (2011) 3D is here: point cloud library (PCL). In: IEEE international conference on robotics and automation (ICRA), May, pp 1–4
28. Izadi S, Kim D, Hilliges O, Molyneaux D, Newcombe R, Kohli P, Shotton J, Hodges S, Freeman D, Davison A, Fitzgibbon A (2011) KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In: ACM symposium on user interface software and technology, October, pp 559–568
29. Sohn K (2014) Ph.D. thesis: optimization of humanoid's motions under multiple constraints in vehicle-handling task. Drexel University, Philadelphia
30. HUBO USA (2014) https://sites.google.com/site/usahubo/project-updates/vehiclemountingwithagolfcart. Accessed 12 Feb 2014
31. Pratt G, Manzo J (2013) The darpa robotics challenge [competitions]. Robot Autom Mag IEEE 20(2):10–12