Task Assignment and Trajectory Optimization for Displaying Stick Figure Animations with Multiple Mobile Robots

Katsu Yamane[1] and Jared Goerner[1]

***Abstract*— This paper presents an offline, centralized motion planning algorithm for displaying stick figure animations by a group of mobile robots equipped with a light source. The algorithm plans collision-free trajectories for the robots such that the figure appears visually consistent across frames including overlaps between body parts. We use 3D motion capture data as input to obtain clean stick figure images. The algorithm consists of three steps: segment generation, robot assignment, and trajectory optimization. In the segment generation step, the input 3D animation is converted to a set of segments of visible robot trajectories in the 2D image plane. The robot assignment step then assigns a robot to each segment using dynamic programming. Finally, the trajectory optimization step computes the complete collision-free trajectory for every robot, including when a robot is not assigned to any segment. We demonstrate the algorithm in simulation using up to 75 robots.**

I. Introduction

This paper presents an offline, centeralized motion planning algorithm for displaying stick figure animations with a group of mobile robots each equipped with a variable-color light source that can also be turned off to make the robot invisible in dark environments. The algorithm plans collision-free trajectories for the robots such that the figure appears visually consistent across frames, meaning that the same body part is represented by the same robot and occluded parts of the body are not visible.

Our current implementation uses 3D motion capture data as input because the algorithm requires clean stick figure images with labels to identify the body parts. We could potentially use hand-drawn 2D animations if we can accurately segment and label the body parts, which is a challenging problem on its own. In this work, therefore, we focus on the motion planning problem assuming that the body parts are correctly labeled.

Figure 1 shows an example of a leaping motion displayed by twenty robots, as they appear to the viewers. Some of the robots, shown as white circles in Fig. 2, are invisible and are repositioning themselves to the next visible locations.

The whole process consists of three steps: segment generation, segment assignment, and trajectory optimization.

The segment generation step generates a set of segments in the image space from the input 3D animations. A *segment* is a trajectory of a body part while it is continuously visible. A segment ends when the body part turns invisible due to occlusion or the number of robots in a limb decreases. In the assignment step, we assign a robot to each segment so that the segment appears consistent throughout its lifetime. If a body part goes invisible and then visible again, however, a different robot may be assigned as a result of the assignment algorithm.

In the next step, each segment is assigned to a robot by applying dynamic programming to find the assignment that minimizes the cost for sharing a robot with multiple segments. The cost function is associated with the expected position errors, wheel velocity, and distances between robots for the most direct transitions between the trajectories. We call a robot "active" if the robot is assigned to one of the given trajectories at a particular time; if not, the robot is "inactive" where the light is turned off and therefore invisible from the viewers.

Finally, the optimization step obtains the collision-free trajectories while the robots are inactive, i.e., not tracking any of the given segments. We apply a standard gradient-descent algorithm with a cost function that includes the wheel speed and distances between robots.

We demonstrate the algorithm in simulation using motion capture data as input and a group of mobile robots with 75 robots.

A. Related Work

Planning coordinated motions for multiple mobile robots has a number of applications from surveillance to art. Our goal is to develop a system for displaying character animations represented as a continuous sequence of stick figure poses. This problem is related to one of the classical problems in multi-robot coordination called formation control.

The major problems related to formation control are: 1) placing robots to appropriately represent the desired image or animation, 2) assigning a robot to each of the desired robot locations, and 3) planning collision-free trajectories to reach the desired locations. Because our focus is in 2), we review the existing work in this area.

A classical approach to task assignment is known as the Hungarian Method as described by Kuhn [1]. While this method gives the optimal assignment, it is not scalable to large number of robots and tasks. Therefore, sub-optimal but scalable algorithms have been developed for large-scale swarm robots, such as the auction algorithm [2]. This algorithm is highly parallelizable and hence scalable to large task assignment problems. These assignment algorithms assume that the number of tasks is equal to the number of robots.

More recently, a number of scalable algorithms have been developed for different applications including formation

[1]Katsu Yamane and Jared Goerner are with Disney Research, Pittsburgh, PA, U.S.A. kyamane@disneyresearch.com

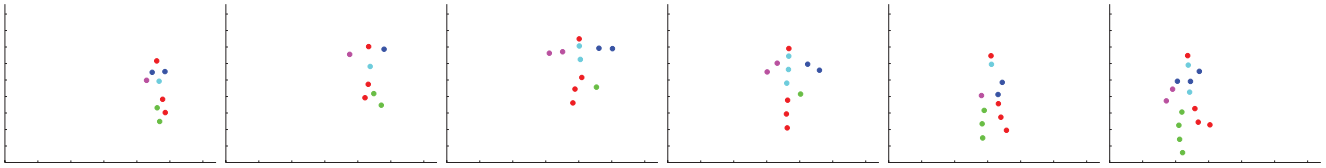

Fig. 1. Stick figure animation represented by a swarm of robots.

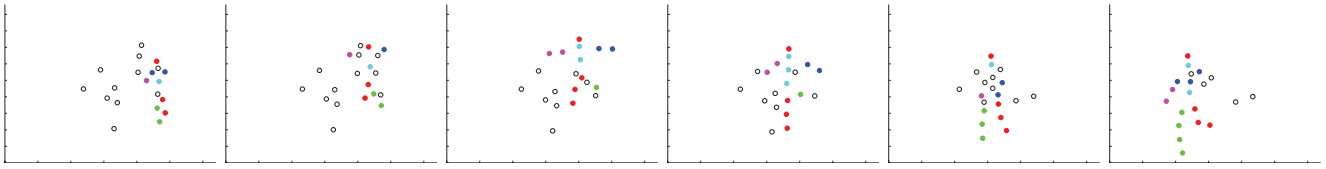

Fig. 2. Same example as Fig. 1 with invisible robots shown in white circles.

control for nonholonomic mobile robots [3], computing non-intersecting trasitions [4], and including the optimization of translation and orientation of the formation at the same time [5].

Unfortunately, we cannot directly apply these algorithms to our problem because some frames in an animation may have fewer points than the number of robots due to occlusions and changes in limb length. Even if the number of points stays the same, we may encounter swaps between robots placed at different parts of the body if we apply the existing assignment algorithms independently to each frame.

General task assignment problems have been discussed at more abstract levels. Most of the recent work in this area focuses on online and/or decentralized algorithms for swarm robots [6], [7], [8]. The problems considered in these papers involve "one-shot" tasks, where each task completes in a short period of time and the transitions between the tasks are not relevant. Our problem, on the other hand, involves tasks that span longer periods of time, i.e., tracking a long trajectory. Also, considering the hardware size and speed limitation, displaying animations will require tight spaces between robots and very fast movements. We therefore choose to perform the planning in an offline and centeralized manner to obtain the best performance.

The most relevant work in terms of the application has been presented by Alonso-Mora et al. [9], where a swarm of mobile robots were used to display images and animations. They first obtain a set of goal positions to maximize the coverage for each given image, and then assign the robots to the goals by the auction algorithm [2]. Their approach may not produce visually consistent results for continuous animations if robots assigned to different parts of the body are swapped between frames.

B. Contribution

Building on the concept of a swarm of small robots as "pixels" [9], we propose a new medium for displaying character animations with physical robots. Motion display has been traditionally done by robots composed of articulated rigid bodies, but it is difficult to design and build robots that

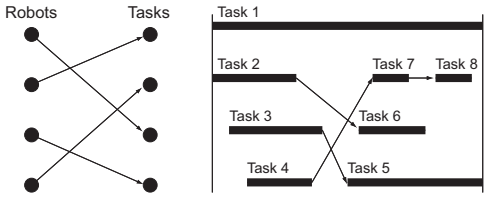

Fig. 3. Conventional (left) and our (right) task assignment problems.

can represent animations with bending and stretching links. Using a swarm of small robots can address this issue.

Technically, our goal is to display a stick figure animation consistently across frames, possibly subject to occlusions by other body parts. In order to realize this goal, we enforce the constraint that each body part, e.g. the right hand, should be represented by the same robot while the part is continuously visible. Therefore, our planning algorithm has to handle a complex problem that involves a set of asynchronous tasks. Furthermore, a robot may be assigned to multiple tasks when possible because the number of tasks may exceed that of the robots. Figure 3 illustrates the difference between conventional task assignment problem and our problem.

II. Hardware Constraints

To incorporate the hardware constraints, we use the parameters of the hardware system used by Alonso-Mora et al. [9]. Each robot has a circular shape with a uniform diameter, and is equipped with a variable-color light source that can be turned off to make the robot invisible from viewers in dark environments. We also assume that the light source can change colors, turn off, or turn on instantaneously. The robots are driven by two differential wheels. We denote the radius of the robots by $r = 0.025$ (m) and the maximum wheel speed at the ground by $v_{max} = 0.5$ (m/s).

The motion planning algorithm works in the image space after the 3D animations are projected onto the image plane. Because the image space is expressed in pixels, we define the spatial scaling factor α (> 0) that converts the physical space in meters to the image space in pixels. The robots' radius in

the image space R, therefore, must satisfy the relationship

$$R = \alpha r. \tag{1}$$

Similarly, V_{max}, the maximum wheel velocity in the image space, is computed by

$$V_{max} = \alpha v_{max}. \tag{2}$$

Because the physical robots may not be able to realize the wheel speed required by the animations, we also define the temporal scaling factor β $(0 < \beta \leq 1)$ to convert the animation time to real time. If we denote the maximum wheel velocity in pixels during the planned robot trajectories by $\hat{V}_{max}$, β should satify

$$\beta \hat{V}_{max} \leq V_{max}. \tag{3}$$

Substituting Eqs.(1) and (2) to Eq.(3), we obtain the following relationship:

$$\beta \leq \frac{v_{max}}{r} \frac{R}{\hat{V}_{max}}. \tag{4}$$

Because we prefer β to be as close as possible to 1, the planner should try to increase R and decrease $\hat{V}_{max}$.

III. Segment Generation

As the source animation, we use 3D animation data reconstructed from human motion capture data. While we should be able to use hand-drawn 2D animation as the source, it is difficult to accurately segment and label the character from such animations without extensive manual intervention. In this paper, therefore, we focus on the motion planning problem by using 3D animation data as input.

The steps to generate the segments are illustrated in Fig. 4. We first project the 3D joint positions onto the image plane of a given viewpoint. The points of each limb on the image plane are then converted to a smooth curve by a Bezier curve using the 2D joint positions as control points, resulting in a set of curves representing the figure shape. The curves are further converted to a set of dots along the curves such that the maximum number of dots across frames does not exceed the number of robots specified by the user.

The dots are placed so that the distance along the curve between neighboring dots becomes constant. We determine the distance such that the given number of robots sufficiently cover the figure at the frame with the maximum total limb length. We also determine R, the radius of the robots in the image space, by computing the minimum distance between dots and multiplying a user-defined factor, which is 0.18 in our implementation.

We handle occlusions by computing the distance between each pair of dots in each frame, and if the distance is smaller than $2R(1+\sigma)$ where σ is the safety margin to account for tracking errors, eliminating the dot with the larger depth from the viewpoint. We use $\sigma = 0.4$ in our implementation.

Each dot is labeled by the limb and the index within the limb. The dots within a limb are numbered from the end of the limb, which means that a new dot will be introduced from inside if the limb stretches. We believe that this indexing scheme produces more consistent appearance of the figure compared to adding dots from the end.

Finally, we generate the segments by connecting the same dot across the frames. We group the dots with the same limb location because it is desirable to represent the same location on a limb with the same robot in order to display the figure consistently across frames.

IV. Segment Assignment

Let M denote the number of robots, N the number of segments, and F the number of frames of the entire motion. Segment n $(1 \leq n \leq N)$ starts at frame s_n $(1 \leq s_n < F)$ and ends at frame f_n $(1 < f_n \leq F,\ s_n < f_n)$. Also without loosing generality, we can order the segments such that $s_i \leq s_j$ holds if $i < j$.

The role of the segment assignment step is to assign a robot to each segment, i.e., find a mapping from segment to robot $\mathcal{A}(n) = m$ $(1 \leq m \leq M)$ for every segment $n = 1, 2, \ldots, N$.

We formulate segment assignment as an optimization problem to minimize a cost function

$$Z_{assign} = \sum Z_{assign}(i,j) \tag{5}$$

where $Z_{assign}(i,j)$ represents the cost when a robot tracks segment i followed by segment j, and has the form

$$Z_{assign}(i,j) = w_p Z_p(i,j) + w_v Z_v(i,j) + w_c Z_c(i,j) \tag{6}$$

where w_p, w_v and w_c are user-defined weights, $Z_p(*)$ is the position error cost, $Z_v(*)$ is the velocity cost, and $Z_c(*)$ is the collision cost.

Obviously, segments i and j cannot overlap in time, i.e., $f_i < s_j$ must hold. For example, if each thick line in the right figure of Fig. 3 represents a segment, $Z_{assign}(2,5)$ is defined but not $Z_{assign}(2,3)$.

The position error cost Z_p represents the robot position error introduced when segments i and j are connected smoothly. In our implementation, we estimate this error assuming that the segments are interpolated by piece-wise cubic b-splines, and control points for the b-splines are placed along the line connecting the end of the segment i and the start of segment j, as shown in Fig. 5. The error term is computed by

$$Z_p(i,j) = ||\boldsymbol{p}_{if} - \hat{\boldsymbol{p}}_{if}||^2 + ||\boldsymbol{p}_{js} - \hat{\boldsymbol{p}}_{js}||^2 \tag{7}$$

where $\boldsymbol{p}_{if}$ and $\boldsymbol{p}_{js}$ are the final and initial positions of segments i and j, and $\hat{\boldsymbol{p}}_{if}$ and $\hat{\boldsymbol{p}}_{js}$ are their interpolated positions respectively.

The velocity cost Z_v is the square of the maximum wheel speed on the interpolated trajectory shown in Fig. 5. At every frame k during the inactive period $(f_i < k < s_j)$, we compute the linear velocity $(v_{kx}\ v_{ky})^T$ and acceleration $(a_{kx}\ \ a_{ky})^T$. Then the left and right wheel speeds are computed by

$$s_{k\pm} = v_k \pm W\omega_k \tag{8}$$

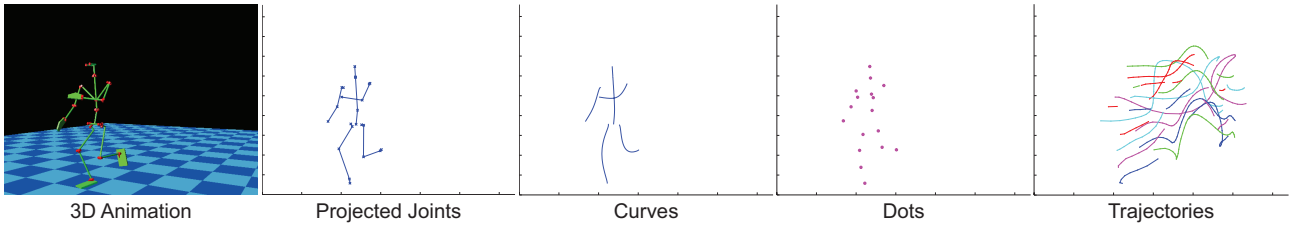

Fig. 4. Generating the trajectories from 3D animations.

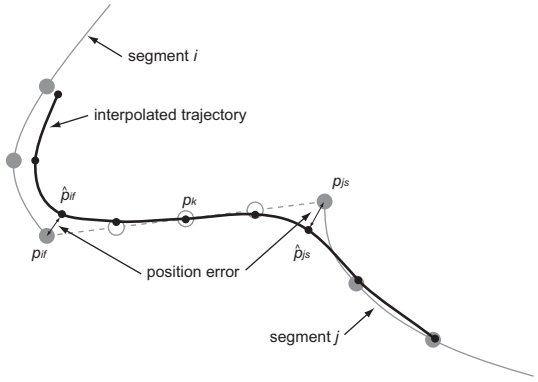

Fig. 5. Connecting two segments.

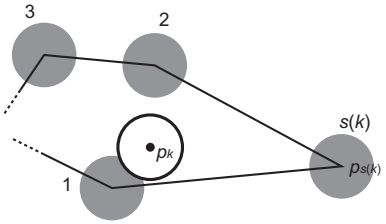

Fig. 6. Computing the collision cost.

where $v_k = \sqrt{v_{kx}^2 + v_{ky}^2}$ and ω_k is the angular velocity of the robot given by

$$\omega_k = \frac{v_{kx}a_{ky} - v_{ky}a_{kx}}{v_k^2}. \tag{9}$$

Using the wheel speeds, $Z_v(i,j)$ is computed by

$$Z_v(i,j) = \max_{k, f_i<k<s_j} (s_{k+}^2 + s_{k-}^2). \tag{10}$$

In our implementation, we evaluate the wheel speeds at an interval of 1/10 frames for more accurate computation of the maximum speed. Also, since the angular velocity in Eq.(9) is not defined when $v_k = 0$, we make sure that no successive control points are given the exact same position.

The collision cost is computed based on the distance from the robots that are actively tracking given segments as follows:

$$Z_c(i,j) = \max_{k, f_i<k<s_j} \exp \ -||\boldsymbol{p}_k - \boldsymbol{p}_{s(k)}|| \tag{11}$$

where $\boldsymbol{p}_k$ is the position of the robot at frame k on the interpolated trajectory shown in Fig. 5, and $\boldsymbol{p}_{s(k)}$ is the position of one of the active robots at frame k, where the index $s(k)$ is obtained as follows.

First sort the active robots at frame k in the ascending order of the distance from $\boldsymbol{p}_k$. Then, starting from $s = 1$, increase s one by one until the polygon formed by $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s$ includes $\boldsymbol{p}_k$ (see Fig. 6). The intuition behind this term is that it would be easier to avoid the active robots if there is more space left to move away from colliding robots.

To find the optimal segment assignment, we apply the dynamic programming algorithm shown in Algorithm 1, where Q is the queue of current possible robot assignments. Function $addNode(*)$ adds a node to the queue with four parameters representing the parent node, trajectory, robot, and cost. Function $getBest()$ returns the node with the smallest cost value. If the best node is associated with the last (N-th) segment, we can terminate the search because all trajectories have been assigned a robot (line 4). Otherwise we first add new nodes associated with the next trajectory as described in the next paragraph (line 7), and then remove the processed node from the queue (line 8).

Function $addDescendants(x)$ adds new nodes associated with the next trajectory. For each robot, we first find the segment that the robot has previously been assigned (line 2). If the robot has never been assigned to a segment before, we add a node representing the assignment of this robot to the segment (line 4) and break. We do not have to process more robots because the remaining robots should have not been assigned to any segment and assigning any of these robots is essentially the same.

If the robot has been assigned to a segment, we first make sure that it does not overlap with the segment to be assigned (line 7). If the transition is feasible, we compute the associated cost (line 8) using the cost function (5) and add a new node (line 9).

The original cost function (6) gives the optimal assignment. If there are many trajectories to track, however, it may not be realistic to obtain the optimal assignment within a reasonable time. In this case, we add another small cost to bias the search towards advancing through the trajectories while giving sub-optimal solution:

$$Z_a = a(N - n) \tag{12}$$

where a is a user-defined constant parameter and n is the index of the last trajectory with a robot assignment. We can adjust a considering the trade-off between computation time

Algorithm 1 Search Optimal Assignment

Require: List of N segments
1: $Q.addNode(NULL, 0, 0, 0)$
2: **while** Q not empty **do**
3: $x \leftarrow Q.getBest()$
4: **if** $x.segment == N$ **then**
5: **return** x
6: **end if**
7: $Q.addDescendants(x)$
8: $Q.remove(x)$
9: **end while**
10: **return** NULL

Algorithm 2 $Q.addDescendants(x)$

1: **for** $m = 1 \dots M$ **do**
2: $n \leftarrow x.prevSegment(m)$
3: **if** n is not a valid segment **then**
4: $Q.addNode(x, x.segment + 1, m, x.cost)$
5: break
6: **else**
7: **if** $f_n < s_{x.segment+1}$ **then**
8: $c \leftarrow cost(n, x.segment + 1)$
9: $Q.addNode(x, x.segment + 1, m, x.cost + c)$
10: **end if**
11: **end if**
12: **end for**

and optimality.

We illustrate the search algorithm for the problem of assigning four robots to the segments shown in the right figure of Fig. 3. The search algorithm builds a tree as shown in Fig. 7 to list all possible robot assignments. Each node of this tree contains the index of the robot assigned, and the depth indicates the segment which the robot is assigned to.

In this example, Algorithm 1 proceeds as follows. Becase Segments 1–4 overlap, the only possible assignment for these four segments is to assign Robots 1–4 respectively, as indicated by the top four nodes in Fig. 3.

We then move on to Segment 5. Because Segment 5 does not overlap with Segments 2–4, we can assign either Robot 2, 3, or 4 as indicated by the three branches between fourth and fifth rows. Figure 7 also indicates the cost function value associated with each branch. For example, the branch from Segment 4 (Robot 4) to Segment 5 (Robot 3) uses the cost to transition from Segment 3 to Segment 5 because Robot 3, the robot assigned to Segment 5, has also been assigned to Segment 3. Note that we do not use $Z_{assign}(4, 5)$ here because a different robot has been assigned to Segment 4.

Let us assume that assigning Robot 4 to Segment 5 is the current best choice returned by $getBest()$. We therefore apply $addDescendants()$ to Robot 4, which results in two descendant nodes Robot 2 and Robot 3 because Segment 6 does not overlap with Segments 2 or 3. The total costs are $Z_{assign}(4, 5)+Z_{assign}(2, 6)$ for Robot 2 and $Z_{assign}(4, 5)+Z_{assign}(3, 6)$ for Robot 3. These costs will be compared to

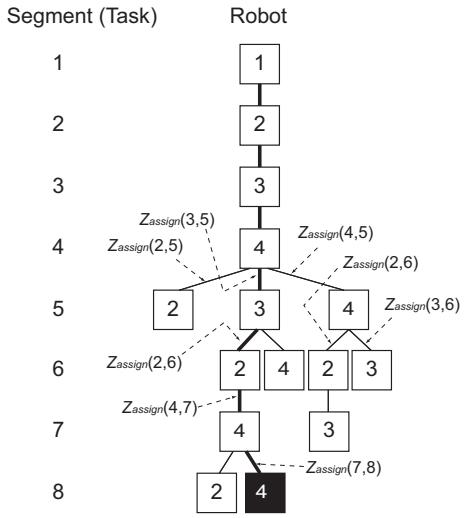

Fig. 7. Illustration of the search algorithm.

$Z_{assign}(2, 5)$ and $Z_{assign}(3, 5)$, the costs of the remaining nodes at Task 5. If $Z_{assign}(3, 5)$ is the smallest among the four, we add Robot 2 and 4 to Robot 3 at Task 5.

We repeat this process until $getBest()$ returns a node that handles the last segment. If we get the node shown in black as a result, we can trace the branches up the tree until we reach the root node as shown in thick black lines. These lines represent the robot assignment shown in Fig. 3.

V. Optimization

We now optimize the inactive robot trajectories to obtain collision-free motions. More specifically, we optimize the control point locations during the inactive periods for the piece-wise cubic b-splines of each robot using the linear interpolation in Fig. 5 as the initual guess.

Note that there are no or very few collisions between active robots because we have already removed the background dots in the segment generation step. It is possible that smoothing of the robot trajectories by cubic b-spline may introduce collisions. However, we consider such collisions when we adjust the spatial and temporal scales after the optimization step. Thus, we will only consider the inactive robot positions in the main optimization.

Another important point of this optimization is that it is not necessarily better to resolve all the collisions in terms of maximizing the temporal scale β, because avoiding collisions by moving around other robots often requires larger wheel speeds that results in smaller temporal scale. We therefore include the collisions as a cost function term instead of hard constraints.

We apply the gradient-descent algorithm to every inactive section independently in the following order:

1) For all robots, optimize the inactive sections between two active sections, in the ascending order of the length of the inactive period.
2) For all robots, optimize the inactive sections at the beginning, in the ascending order of the length of the

inactive period.

3) For all robots, optimize the inactive sections at the end, in the ascending order of the length of the inactive period.

The idea is to start from the most restrictive part of the trajectories. The motion between active trajectories is more restrictive because both ends are constrained. Also, we would not be able to move the control points much if there are fewer inactive frames.

Each optimization attempts to minimize the following cost function:

$$Z_{optim} = w_{op}Z_{op} + w_{ov}Z_{ov} + w_{oc}Z_{oc} \tag{13}$$

where, similarly to Eq.(5), the three terms represent the position error, velocity cost, and collision cost.

The position error cost is simply the sum of the squared distances between the original and modified control points. The velocity cost Z_{ov} is also similar to Z_v, except that we use the sum of the squared wheel speeds at all inactive frames instead of the maximum.

The collision cost is also similar to Z_c but we use a slightly different formulation because we do not have to bring two robots farther apart once the collision is resolved. The collision cost between two robots separated by distance d is

$$z_c(d) = \begin{cases} (d - 2R(1+\sigma))^2 & \text{if } d < 2R(1+\sigma) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where $\sigma \geq 0$ is a safety margin.

Because the optimization does not guarantee complete collision avoidance, we recompute the robot radius by taking the minimum distance between the robots during the planned motion and dividing it by 2. We then use Eq.(4) to recompute the temporal scaling factor β.

VI. Results

We demonstrate the algorithm using 30–75 robots in simulation and two motion capture sequences selected from Carnegie Mellon University Motion Capture Library [10]. Please refer to the accompanying video for the animations.

Figure 8 shows some snapshots from the original 3D animation and generated robot motions using 30, 45, 60, and 75 robots. Figure 9 compares two results obtained from the sets of parameters marked with ^{*1} and ^{*2} in Table I. While Fig. 9 shows the corresponding frames of the two motions, the video shows them four times faster than the corresponding real speed based on their temporal scales. Due to the difference in the temporal scales, the video clips have different durations.

Tables I–III give more quantitative results.

Table I compares the temporal scale β obtained for 75 robots with different assignment and optimization parameters. This result shows that there is a specific set of parameters that gives the best (largest) temporal scaling.

Table II compares the maximum possible robot radius R to prevent collisions. For the same physical robot size, the actual size of the field to cover the image decreases as the radius increases.

TABLE I

Temporal scale β for various assignment and optimization parameters (common parameters: $M = 75$, $w_c = 1$, $w_v = 10^{-6}$, $a = 0.05$, $w_{op} = 10^{-3}$, $w_{oc} = 1$)

		w_p		
		10^{-1}	10^{-2}	10^{-3}
w_{ov}	10^{-7}	0.0259	0.0270	0.0187
	10^{-8}	0.0209	0.0286^{*1}	0.0230
	10^{-9}	0.0269	0.0215	0.0174^{*2}

TABLE II

Robot radius R (pixels) for various assignment and optimization parameters (common parameters: $M = 75$, $w_c = 1$, $w_v = 10^{-6}$, $a = 0.05$, $w_{op} = 10^{-3}$, $w_{oc} = 1$)

		w_p		
		10^{-1}	10^{-2}	10^{-3}
w_{ov}	10^{-7}	1.58	1.66	1.14
	10^{-8}	1.28	1.77	1.43
	10^{-9}	1.65	1.65	1.43

Table III compares the maximum wheel speed $\hat{V}_{max}$ found in all robot trajectories. As expected, we obtain smaller $\hat{V}_{max}$ when we increase the velocity cost weight w_{ov}. Interestingly however, smaller w_p which in turn should emphasize smaller wheel speed in segment assignment, tends to result in larger wheel speeds after optimization. We speculate this is because larger w_p tends to let the assignment choose transitions that do not require abrupt turns, which also leads to smaller wheel speeds.

Finally, Fig. 10 shows an example generated from a different motion (walking). In this example, we placed the viewpoint so that occlusions happen frequently. As a result, interesting behaviors emerge from the planning to handle occluded limbs as shown in Fig. 11. In the left example, the robots that had been forming the occluded limb pass between those forming the foreground limb. In the right example, some of the robots are left behind and swapped with other robots that are waiting for the limb to be visible again.

All the results have been generated using the parameters marked by ^{*1} in Table I unless otherwise noted.

VII. Conclusion

In this paper, we presented a method for displaying stick figure animations with multiple mobile robots. The main technical contribution is the task assignment algorithm for maintaining consistent appearance of the figure throughout

TABLE III

Maximum wheel speed $\hat{V}_{max}$ ($\times 10^3$ pixels/s) for various assignment and optimization parameters (common parameters: $M = 75$, $w_c = 1$, $w_v = 10^{-6}$, $a = 0.05$, $w_{op} = 10^{-3}$, $w_{oc} = 1$)

		w_p		
		10^{-1}	10^{-2}	10^{-3}
w_{ov}	10^{-7}	1.23	1.23	1.23
	10^{-8}	1.23	1.23	1.24
	10^{-9}	1.23	1.53	1.95

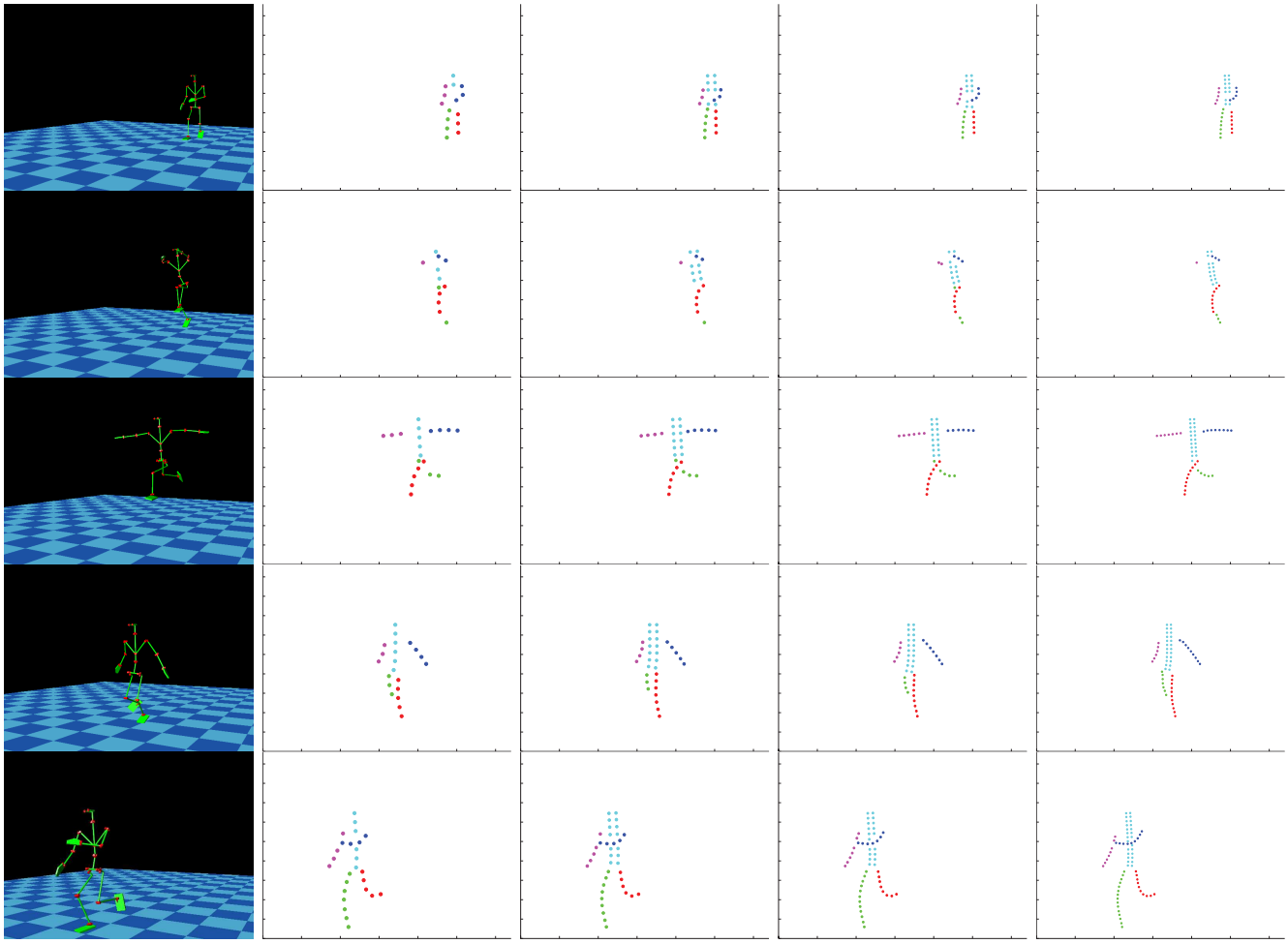

Fig. 8. From left column: original 3D animation, 30 robots, 45 robots, 60 robots, and 75 robots.

an animation. For this purpose, we developed a task assignment algorithm based on dynamic programming.

We demonstrated the algorithm in simulation using 30 to 75 robots displaying two 2D stick figure animations generated from motion capture data. The results showed that our algorithm can display stick figure animations with significant limb length changes and occlusions.

For future work, we would like to consider online local collision avoidance to cope with tracking errors encountered when the planned trajectories are executed on hardware. Another interesting direction would be online planning for interactive applications.

ACKNOWLEDGMENT

The authors would like to thank Paul Beardsley and Javier Alonso-Mora at Diseny Research, Zurich for their valuable comments during the preparation of the manuscript.

REFERENCES

[1] H. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, pp. 83–97, 1955.

[2] D. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, pp. 105–123, 1988.

[3] J. Desai, J. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, 2001.

[4] M. Broucke, "Disjoint path algorithms for planar reconfiguration of identital vehicles," in *Proceedings of the American Control Conference*, vol. 3, 2003, pp. 2199–2204.

[5] M. Ji, S. Azuma, and M. Egerstedt, "Role assignment in multi-agent coordination," *International Journal of Assistive Robotics and Mechatronics*, vol. 7, no. 1, pp. 32–40, 2006.

[6] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms," in *Proceedings of Robotics: Science and Systems*, 2005.

[7] N. Michael, M. Zavlanos, V. Kumar, and G. Pappas, "Distributed multi-robot task assignment and formation control," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 128–133.

[8] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithm design for multi-robot generalized task assignment problem," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 4765–4771.

[9] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart, and P. Beardsley, "Image and animation display with multiple mobile robots," *The International Journal of Robotics Research*, vol. 31, no. 6, pp. 753–773, May 2012.

[10] "Carnegie Mellon University Graphics Lab Motion Capture Database," http://mocap.cs.cmu.edu/.

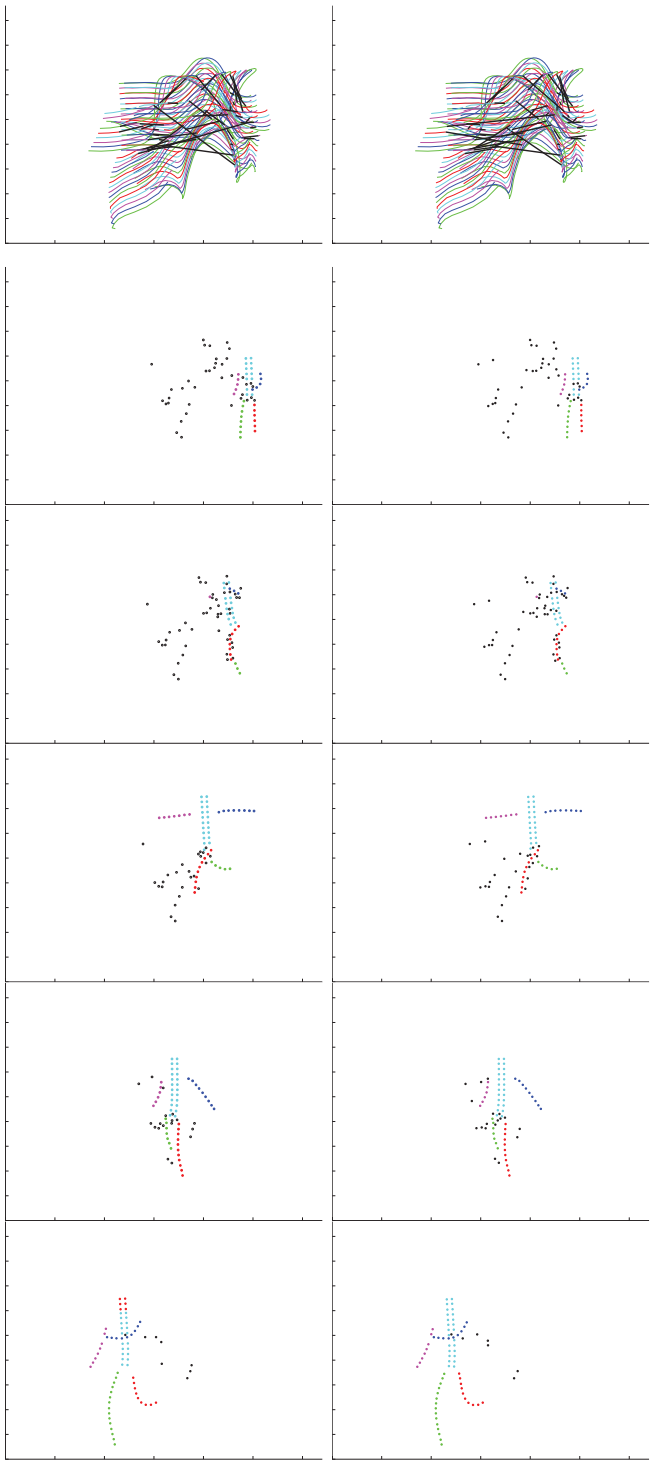

Fig. 9. Planned motions obtained from the sets of parameters marked by $*^1$ (left column) and $*^2$ (right column) in Table I, shown with the invisible robots. The top row depicts the segment assignments where the thick lines represent transitions between segments of the same robot.

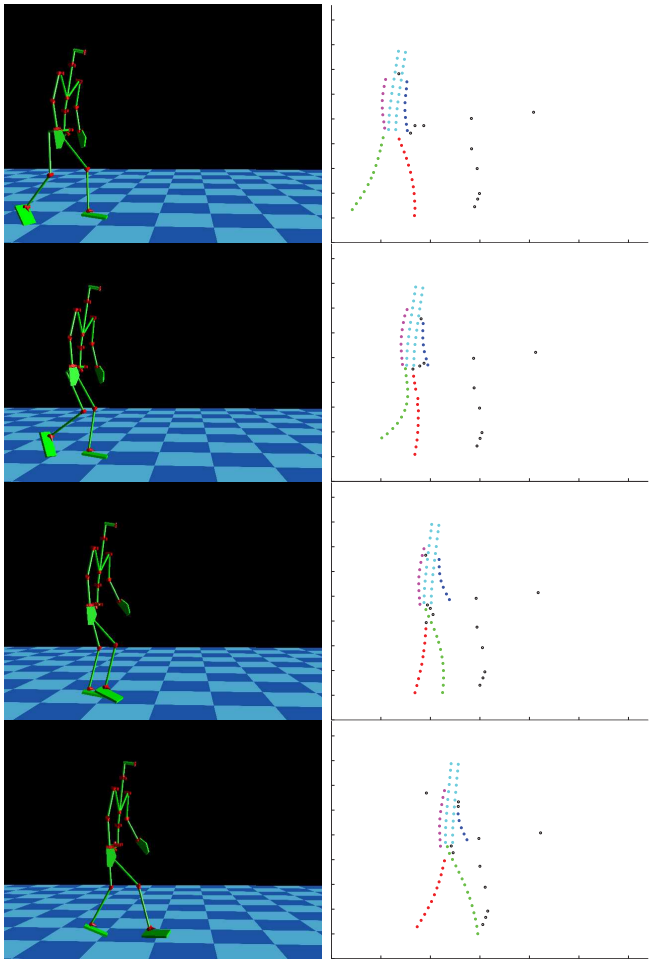

Fig. 10. Walk example.

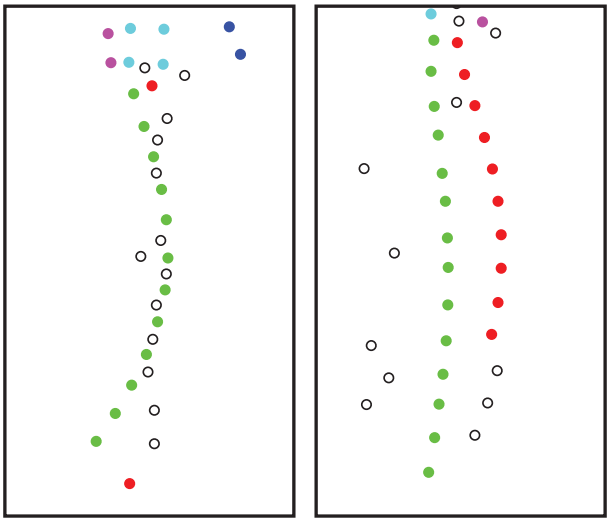

Fig. 11. Handling occluded limbs; left: passing between active robots, right: waiting for the limb to be visible.