Situational Awareness for Manufacturing Applications

Olivier St-Martin Cormier, Andrew Phan and Frank P. Ferrie
Dept. Electrical and Computer Engineering &
McGill Centre for Intelligent Machines
Montreal, CANADA
Email: {olivier,aphan2,ferrie}@cim.mcgill.ca

***Abstract*—Collaboration between human workers and robotic assistants is seen as one way to increase both flexibility and efficiency in a production line environment. In this setup, human workers can be assigned tasks that require high perceptual ability, dexterity and judgement, supplemented by robotic assistants that can perform work of low (skill) value, such as fetching and delivering parts and tools. Key to such a strategy is the ability of the automated system to maintain total awareness of the states of all key players (humans, robots, machinery, parts) and take the necessary action to carry out the manufacturing while maintaining the safety of the human workers. We refer to this attribute as Situational Awareness, and in this paper present both an implementation and a case study in the form of a system that tracks the articulated 3D pose of a group of human workers in an enclosed area.**

***Keywords*-sensors, networks, databases, real-time**

I. Introduction

Robotic systems have long played an important role in modern manufacturing with even predictions of workerless factories dating back to Diebold in the 1950's [1]. Although automation has had a profound impact, human workers still play important roles, especially in an environment where flexibility is a key factor, e.g. automotive assembly line. An emerging strategy in recent years has been to augment human workers with assistive devices or even robot assistants [2], [3]. This has the advantage of retaining high skill level and flexibility, while off-loading work of lesser value to automation. There are a number of important considerations here. A robot assistant must take direction as well as communicate/coordinate with its human counterpart in performing a task, a problem that has been receiving a lot of attention in the human-robot interaction (HRI) community [4], [2], [5]. Further, the safety of the human worker and the surrounding plant must be accounted for, e.g., by incorporating force limits into the design of robotic systems (safe robots) [6], and/or identification of potential hazards [7], [8].

Underpinning these considerations is the need for a robot assistant to be aware of its own state, that of its human co-worker, the state of the task being performed, as well as the state of the larger environment in which it operates. States of different entities are determined through processes of inference, operating at the lowest levels on data provided by sensors that encode different properties of the environment, and at higher levels on inferences generated by other entities operating within the robotic assistant's enviromnent. Collectively we refer to the information provided by these processes of inference as *Situational Awareness* (SA). Although particular details can vary, one can abstract SA as a common, unified representation of the environment that makes explicit what needs to be known to perform the required tasks.

Implementation of SA in the form of a system raises a number of constraints that arise from the requirements of SA consumers in the human robot collaboration.

1) Unified Representation: Information should be available from a single source, in real-time without explicit knowledge of where that information is located (e.g. publish-subscribe architecture).
2) Asynchronous Access: Information should be available on an anytime basis, with timescale and latency requirements imposed by the client process.
3) Sensor/Data Fusion: Information from sensor/data sources operating at different locations and timescales should be integrated by the system and be presented as a whole (retaining access to all source data).
4) Consistency Across Representations: Where possible, ambiguous or incorrect results due to sensor noise, ambiguity in the scene, or incorrect assumptions in a particular algorithm, should be handled as part of the SA system.

The case study presented here is that of a system that monitors the precise body pose of one or more human workers collaborating with a robot assistant in a production line environment. A sensor network composed of 4 Kinect 2.5D cameras continuously samples the workspace (Figure 1) at a rate of approximately 14Hz. This information is fused into a single representation, from which any process, such as the 3D Human Pose Tracker (Section III), can subscribe[1]. From this information the tracker computes an articulated, 3D skeletal model with 32 degrees of freedom (dof), and updates the current description in a globally accessible, real-time database - Situational Awareness Database (SADB) (Section II). The monitoring process can access the SADB, asynchronously and obtain a best estimate of each occupant in the workspace.

[1]Transmitting the raw RGB-D images from 4 Kinects over the network maxed out a gigabit port with a max frame rate of 12.5Hz. As a result, for the live system, the raw Kinect RGB-D data are read directly from the cameras instead of going through the SADB because it allows us to exceed 12.5Hz limit while minimizing overhead/latencies. We only pushed the raw RGB-D images into the SADB for the purpose of recording the validation data.

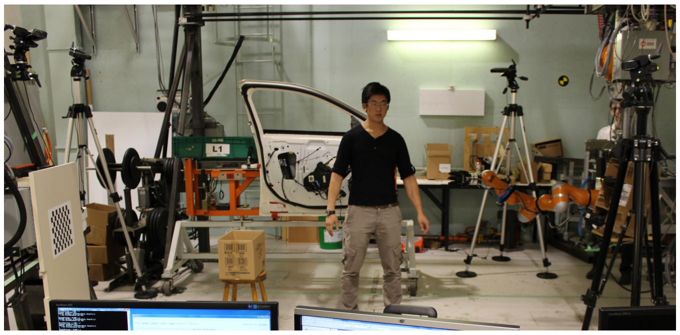
Figure 1. Workspace with 4 Kinects

The constraints described above are enforced as follows. Unified representation is provided by the SADB, operating at video frame rates and implementing a blackboard architecture in which all entities can share information. Sensor information can be written by one process and accessed by another with minimal latency. Consumers of sensor information operate on this data and write back results which in can turn be accessed by other processes. Asynchronous access is handled by SADB, but anytime access is handled by processes responsible for maitaining the representations. Another process operates on the 4 Kinect datastreams and maps each into a common frame of reference. The resulting voxel space is continually updated at sensor frame rates. Consistency enforcement is implemented at the level of the 3D skeletal model, where optical flow and soon range flow are used to resolve errors in limb identification caused by contact problems.

The remainder of the paper elaborates further on SADB in Section II and the 3D Human Pose Tracker in Section III. A case study of the resulting system is presented next in Section IV with experimental results describing the performance of the system. Finally, the paper concludes in Section V with a summary of the paper and lessons learned.

II. The Situational Awareness Database

The SADB system has been designed from the ground up to fulfill the requirements stated earlier. This section will focus on various aspects of SADB related to these requirements.

While designing the database, we looked at the original blackboard implementation by Erman et.al. [9] and other classic descriptions of blackboard architectures [10], [11]. Modern database systems were also studied to refine the requirements and were also compared to determine whether to use a Structure Query Language (SQL) [12], [13] or a NoSQL [14], [15], [16] interface to the database. The intent of SADB is to be as flexible as traditional blackboard systems while presenting a modern interface to clients.

A. System Organization

The primary goal of SADB is to provide a centralized location to store and organize contextual knowledge related to an observed system. The client-server topology was adopted so that the only required prior knowledge to connect to the database is the IP address or hostname of the server.[2]

To simplify interactions with the database, SADB was built as NoSQL database. This means that most database functions are presented to clients as a convenient object oriented Application Programming Interface (API). This removes the need to format request strings for most database operations. Implementation-wise, the SADB system was written in C++ and has been successfully tested and used on Linux and Windows platforms. The API currently provides the functions for the 37 distinct commands supported by the SADB server shown in Table I. The description of each command is omitted because of space constraints, but most of the functions names should be self explanatory.

Table I
List of supported commands

0x01: Ping	**0x02**: Create Object
0x03: Delete Object	**0x04**: Set Object Name
0x05: Get Object Name	**0x06**: Set Object Description
0x07: Get Object Description	**0x08**: Get Object by Name
0x09: Send Object	**0x0A**: Get Latest Object Value
0x0B: Get Object Value at Nearest Timestamp	**0x0C**: Get Object Value at Timestamp
0x0D: Reset Blackboard	**0x0E**: Create Category
0x0F: Delete Category	**0x10**: Set Category Name
0x11: Get Category Name	**0x12**: Set Category Description
0x13: Get Category Description	**0x14**: Get Category by Name
0x15: Add Object to Category	**0x16**: Remove Object from Category
0x17: Find Object	**0x18**: Remove Object's Oldest Value
0x19: Keep n Latest Object Values	**0x1A**: Get Latest Object Timestamp
0x1B: Get Object Value at Next Timestamp	**0x1C**: Get Object Value at Previous Timestamp
0x1D: Get Number of Objects	**0x1E**: Get Number of Categories
0x20: Is Object ID Valid?	**0x21**: Is Category ID Valid?
0x22: Get Next Timestamp	**0x23**: Get Previous Timestamp
0x24: Get Nearest Timestamp	**0x25**: Create Trigger Program

The data on the database is structured in chunks called objects. Each object is defined as a container to store multiple measurements of a single piece of information about the environment. Each measurement is composed of a discrete timestamp and data of arbitrary length and dimensions. No restriction is imposed on the type of data. Objects also store some metadata such as the name, description, and a unique identification code.

For most database operations, objects are indexed by their unique ID code, which can be transparently obtained from the name of the object. To further organize the data stored on the blackboard, objects can be classified into categories. The categorization of the data is similar to the organization of Redis [15] and the file system presented by De Chiara

[2]In the initial implementation reported here, a side process was also used to retrieve source data which required access to named network sockets.

et.al.[17]

When classified properly, objects stored in the database can be found by emitting object queries. These queries use mathematical set operations to locate objects that match any desired conditions. Currently, the union, intersection and complement functions are implemented. By combining these three operations, arbitrarily complex query operations can be constructed.

The rationale behind the categorized data organization is that any client connecting to the system can interact with the data without knowing in advance the names or number of objects present. This organization also helps in making the database more flexible by allowing a classification of data based on a semantic structure.

B. *Real-time operation*

To ensure real-time operation of the database system, the transfer protocol was designed as a representational state transfer (REST) protocol, conceptually very similar to the hyper text transfer protocol (HTTP). Each request from clients is sent as a series of packets beginning with a header followed by a payload. The header includes the opcode of the requested operation, the ID of the object that is targetted and other command-related parameters. The entire network-related aspect of SADB is handled by the API and is thus invisible to users of the system.

To further minimize data access time, the entire database is kept in system memory. This alleviates disk read and write latencies. While this design decision obviously imposes a hardware requirement on the amount of system memory, multiple features have been implemented to reduce this problem. The client API includes functions to discard historical data based on a threshold or to keep only a given number of values. Clients are thus partly responsible for managing their usage of memory. To prevent the risk of data losses associated with system restarts, the possibility of saving and restoring the entire content of the database to disk has been implemented.

As with most database systems, the performance is tightly coupled to the speed of the network it operates on. After careful experiments, it was determined that the transfer frequency decreases linearly with the network speed, as expected. Both read and write performance have been shown through experiments to be real-time, given appropriate network infrastructure.

C. *Asynchronous Access*

As mentioned earlier, each object value stored on the database includes a timestamp. These timestamps are used to index data within an object based on measurement time. Each timestamp is defined as an integer amount of seconds since the unix epoch and the number of milliseconds since the previous integer second.

As the sensors and clients measuring and generating data do not all run at the same rate and do not necessarily have the same latency characteristics, the task of timestamping the data was left to the clients. This allows a client to set the timestamp of a value closer to the measurement time and thus negates the effect of processing and network transmission delays on the stored timestamps. A local network time protocol (NTP) server is used to synchronize the SADB server and all clients to the same clock.

The SADB protocol includes five different functions to access object values. Clients can choose to request data using any of the five methods independently for each request.

- **Get Latest Value:** This function fetches the latest value stored for an object from the database. This is the fastest and simplest method.
- **Get Nearest Value:** This function returns the value at the stored timestamp closest to a target time. This method allows the query of historical values.
- **Get Next/Previous Value:** These two functions are used to step through the stored values in chronological order by returning the value at the timestamp after or before a target time.
- **Get Value:** This function also allows access to historical value, but will compute an approximation of the value at the requested timestamp by using interpolation. Extrapolation can also be used to generate future values. Linear and polynomial interpolation algorithms are available. Parameters such as the degree of the polynomial and the number of values to be considered for interpolation are configured by the request parameters and can vary between requests. Implementing the interpolation functions on the server greatly reduces the amount of data transferred over the network that would be required for clients to generate their own approximations. The only requirement of the interpolation methods is that the dimensions of the object values being used for the approximation need to be the same.

This asynchroneous access to data removes the needs for process synchronization between the clients, thus making the overall system more flexible. Clients can request the values at whichever time they require, regardless of the source of the data.

D. *Database Management*

To simplify the task of managing the database, a web-based interface through an integrated HTTP server was added to present a human-friendly way to interact with SADB. The web interface includes the ability to view properties and contents of the objects stored on the database as well as explore the categories and view their members. An administration page is also present to allow direct control of the database backup and restore procedure. The database system can also be reset from the web interface.

This ability for humans to interact with the system and view information related to the current state of the world representation can help in increasing the trust between the users and the system.

E. *Comparative Results*

The read and write performance of SADB was compared to the performance of the Redis database system [15] and the MongoDB system [16]. Those two database systems were chosen for comparison because both are NoSQL systems and both are widely used projects. Redis has also been chosen because it is a RAM-based database system. Table II provides the average read and write times measured for the three tested systems. To simulate a plausible system, 16 objects of 1 kilobyte were transfered to and from the server a thousand times to measure the average. To test different network configurations, the test was executed in two different scenarios. First the measures were taken when the client and the server resided on the same machine, then the same test was executed with the server on a different computer.

Table II
AVERAGE READ AND WRITE TIMES FOR 3 DATABASE SYSTEMS

DB System	**Local** (μs) Read	Write	**Network** (μs) Read	Write
SADB	1321	942	10095	11668
Redis	932	1073	10118	11967
MongoDB	348812	2927	3742027	11375

The results support the idea that RAM based databases have better read performance as no disk access is required. The similary in write performance can most likely be attributed to caching prior to writing to the hard drive. It can be seen that the addition of purpose-specific components has no performance impact with respect to other state of the art general purpose database systems.

III. 3D Human Pose Tracker

From a high level, the 3D human pose tracker can be seen as a SA process that abstracts independent, low-level Kinect RGB-D data and produces one or more skeletons that it makes available to all other SA processes or entities such as the robotic assistant. In the proposed collaborative manufacturing application, the 3D human pose tracker is designed to meet two requirements. First, it must provide reliable and fast 3D human pose in the form of a 32-dof skeleton so that the robotic assistant can safely operate alongside one or more human workers. Second, it must adapt to the varying conditions typically encountered on an assembly line such as the variability in worker shapes and size, equipment layout, appearance of all entities and events (i.e. the unpredictable comings and goings of all entities).

To meet these requirements, we used a camera network composed of 4 calibrated Kinect 2.5D cameras placed in an approximately orthogonal configuration. Each Kinect provides RGB and depth images at approximately 14Hz that can be fused to produced a single point cloud for 3D-based algorithms or processed independently using conventional image-based algorithms. We use a camera network designed to maximize observability and minimize occlusions in order to compensate for the minimal amount of priors required by the system. We assume that workers are not missing any limbs and that they adopt an unambiguous T-pose to initialize the tracking.

Estimating reliable 3D human pose is complicated by three types of contacts, referred to collectively as the contact problem: *subject-subject* contacts (e.g. handshake), *subject-object* contacts (e.g. part manipulation) and *self-contacts* (e.g. hand on thigh). These contacts hinder the localization of extremities such as the hands and feet which impact the final skeleton estimate.

In a previous work [18], we proposed a data-driven pipeline that specifically addresses the self-contact problem. The pipeline, implemented in C++ and making use of the GPU for the computationally intensive optical flow image processing algorithm, fused the available RGB and depth data and was optimized to maximize performance while minimizing latencies. Unfortunately, it had two major limitations. First, it did not address the other two types of contacts and, second, errors in the estimated skeleton tended to accumulate because of the always active feedback loop introduced to resolve the self-contact problem. In this work, we propose a revised pipeline (Figure 2) that addresses these two limitations by handling all three types of contacts with the addition of a second corrective feedback loop and by switching to a detection-based approach to determine if it is necessary to branch into either feedback loop.

As a pre-requisite, we first calibrate the camera network using a checkerboard pattern and standard camera calibration techniques [19], [20]. Specifically, for each Kinect, we compute first the intrinsic calibration parameters of each IR and RGB camera while blocking the IR emitter. Then we unblock the IR emitter and compute the stereo calibration parameters between the IR and RGB cameras of each Kinect and between the RGB cameras of Kinect pairs. Finally, we place a large checkerboard at the workspace origin and compute the extrinsic calibration parameters.

A. The pipeline

When the system is first powered on, we start by modelling each depth image background using the Minimum Background algorithm [21] and segmenting the foreground to reduce the number of unnecessary points to project into the workspace reference frame. Using the calibration parameters computed previously, each camera produces an individual foreground point cloud which we merge into a single registered foreground point cloud. Then we downsample the data using a voxel grid, cluster the foreground into blobs using Euclidean clustering [22] and compute features for each blob such as the three inertial principal axis lengths and the convex hull volume [23]. Although the principal axes of inertia are more suited for rigid 3D bodies, we found them to be well suited for distinguishing between human blobs and other blobs in the scene, especially during initialization of the tracker when subjects adopt the T-pose.

Next, we convert each human blob's point cloud to a graphical model called a geodesic distance graph (GDG), where each point corresponds to a node and neighbouring nodes are connected by edges whose weights represent the Euclidean distance between the two neighbours. Then we apply the AGEX geodesic extrema extraction algorithm in [24] to locate the five extrema corresponding to the head, hands and feet. If no previous pose estimate is available, then the labelling of primary landmarks is accomplished by having the subject adopt an unambiguous T-pose with arms straight out to the side and slightly offset towards the front facing direction for reliable left-right disambiguation.

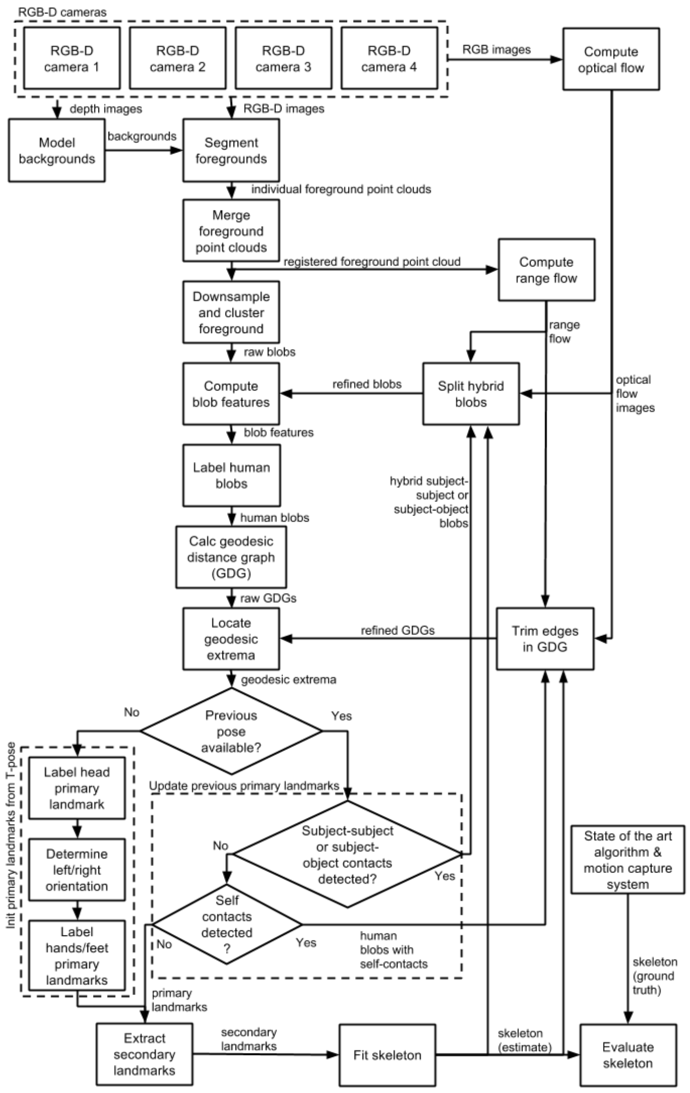

Figure 2. Human tracker data flow diagram

If successful, then we also store the subject's personal characteristics such as the geodesic distance between any two limbs. If a previous pose estimate is available, then we need to check for undesired contacts and take corrective actions, if applicable.

The geodesic distance (GD) is a useful metric because it is pose invariant. Assuming no significant occlusion that would cause the worker to be poorly sampled by the camera network, the GD between two limb ends only changes significantly if there are invalid edges in the geodesic distance graph due to the contact problem. In our revised pipeline, we propose a two-level detection-based consistency check and corrective feedback loop to deal with the problem.

In the first level, we check for human blobs that have suddenly merged with other subjects or objects producing hybrid subject-subject or subject-object blobs that need to be split. Depending on the size of the other subject or object, such blobs will exhibit abrupt changes in convex

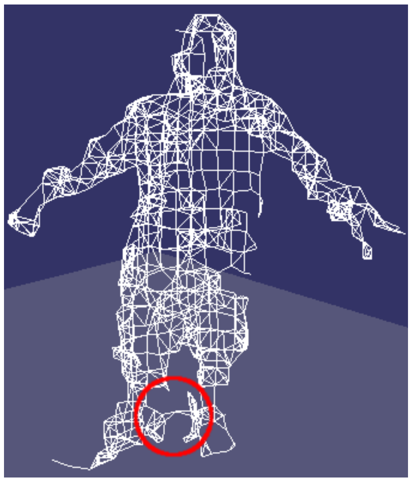

Figure 3. Example geodesic distance graph with an invalid edge highlighted near the feet due to the self-contact problem

hull volume, centroid position, number of point samples, inertial principal axes lengths and orientation, mesh shape or geodesic extrema positions. In addition, changes in the number of blobs from one frame to the next frame, such as a missing blob, also suggests that a hybrid blob may have formed due to the contact problem, provided that no blobs were located near the observation boundary of the camera network in the previous frame. As such, we must split the hybrid subject-subject or subject-object blobs.

In the second level, we use the pose-invariance property of the GD metric and check for self-contacts by verifying the constancy of geodesic distances between all extrema pairs. If a limb fails this check, then there are invalid edges in the GDG that connect two non-adjacent segments (Figure 3). In this case, we must remove the invalid edges in the GDG and re-run the AGEX algorithm in order to locate the desired primary landmarks.

In both cases, we start with the same initial step and use the skeleton estimated in the previous frame to label each point of the blob in the current frame with a most likely blob ID (first level) or body part label (second level). Inspired by the optical flow based 2.5D GDG edge trimming algorithm in [25], we use a 3D blob point labelling algorithm [18] that uses optical flow with a multi-view voting scheme to resolve ambiguities and compensate for the latency of the human tracking system. Specifically, the optical flow images from each camera view serve to estimate the current location of the each previous blob's points. As such, each view contributes up to one vote in determining the label of a given point. We are currently working on enhancing the voting scheme further by incorporating depth data and computing the range flow alongside the optical flow. The main difference after the initial labelling step is that in the first level the labels are used to split hybrid blobs while in the second level they are used to trim edges that connect non-adjacent segments.

At this point, we have located the five primary landmarks corresponding to the head, hands and feet. Next, we obtain additional secondary landmarks, corresponding to the wrists, elbows, shoulders, neck, ankles, knees and hips, by computing the centroid of the locus of points at a preset (or predetermined during the T-pose initialization) geodesic distance offset from the primary landmarks. Finally, the skeleton is fitted to both primary and secondary landmarks using the Levenberg-Marquart non-linear least squared minimization algorithm as in [25] with additional cost terms such as penalizing segment intersections. The estimated pose is then stored in the SADB to be retrieved by other entities in the scene, namely the robotic assistant. Additionally, to ensure that the tracker performs online at interactive rates, the pipeline is implemented in C++ and multi-threaded such that computationally expensive processing modules (e.g. the GDG construction and the skeleton fitting) are computed in parallel in separate threads for each human blob.

B. *Comparison to the Vicon motion capture system*

To validate the human tracking component quantitatively, we compare the estimated skeleton to the commercial marker-based Vicon motion capture system (Figures 4 and 5). As the work described in this paper is still in development, we present preliminary results that show a median error of 0.149 ± 0.082 m for when the worker adopts an ideal pose, a mostly static unambiguous T-pose. Note that this error also includes a constant per-segment bias error corresponding to approximately half the width of body part at the joint because the skeleton provided by the Vicon Nexus software currently only produces a skeleton that is on the surface of the body rather inside the body, at the center of each segment.

The results are promising because they indicate that we are able to approximate the accuracy and performance of an expensive and intrusive marker-based motion capture system using commodity hardware. Furthermore, the Vicon is not immune to the contact problem and exhibits difficulty tracking when markers from other subjects, objects or segments get too close and interfere with each other. Using both intensity and depth data, we are optimistic that the proposed pipeline and feedback loops will address the contact problem and advance the state of the art in markerless, multi-view and multi-subject pose estimation and tracking.

IV. Case Study: Tracking Articulated 3D Pose

The current section will show how the proposed system has been successfully applied to the problem of collaborative work between human workers and robotic assistant in a factory environment.

The robotic assistant operates based on an internal finite state machine, which determines what the current task of the robot is and what conditions must be met before transitioning to the next task. The conditions can either be internal to the robot or measured by external sensors and accessible through SADB. Internal conditions may include the position of the robot or whether a part is still in the robot gripper. The external conditions defined in the project were mostly dependent on the state of the human worker.

Some of the values commonly accessed by the robot were:

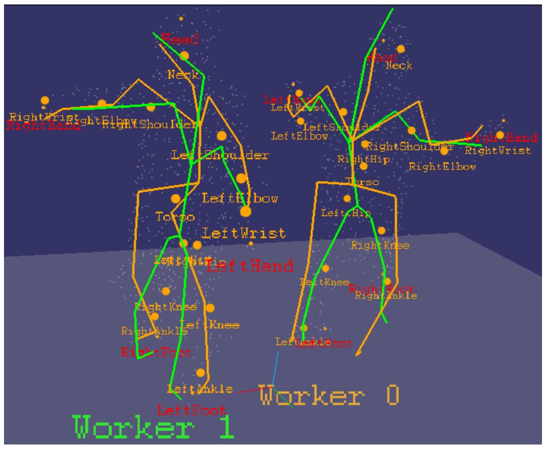

Figure 4. 3D human tracking skeleton estimate (green) compared to Vicon ground truth (orange) for two workers

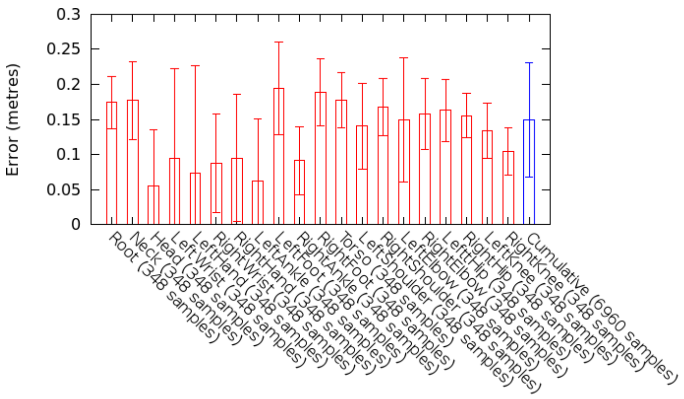

Figure 5. 3D human tracking skeleton median error compared to the Vicon

- **Worker Position:** The position of the worker was often used to determine what the worker was currently doing. If the worker resides within a given distance of the work piece, work is most likely currently being done on that piece. When the worker steps away, the robot may assume that the worker has completed a task.
- **Worker Pose:** To prevent collisions between the robot and the worker, the robot would verify the pose of the human to plan motions.
- **Worker Gesture:** The worker may indicate certain conditions directly to the robot by body gestures such as raising an arm to signal a potential problem or extending an arm toward the robot to request a new part or a tool.
- **Handover Location:** This position was used to determine the best position to place a part so that it could be easily accessible to the worker. This was usually computed as an offset from the hand of the worker.
- **Task State:** The worker's pose was used to infer the state of the task according to the position of the hands. Flags such as Worker Working on Workpiece, Worker Done Working on Workpiece, or Part Disposed provided key input to the finite state machine.

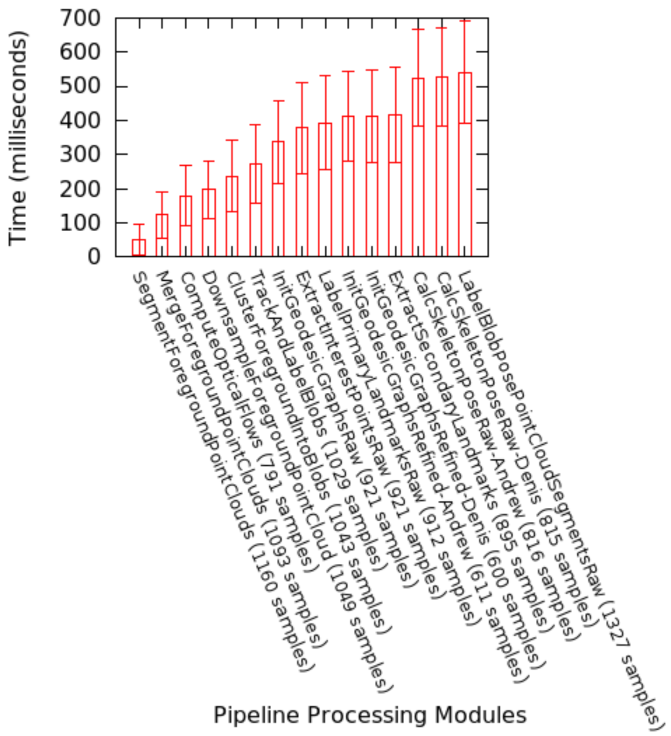

Figure 6. Median cumulative latencies of the 3D human pose tracking pipeline

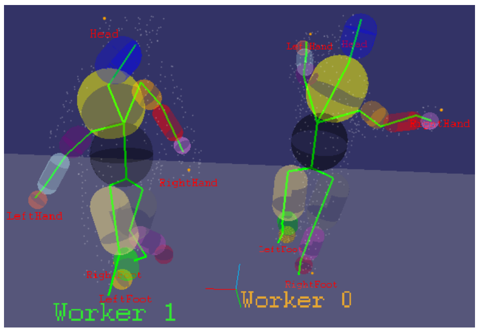

Figure 7. Renderer displays two workers with skeleton and segment shapes

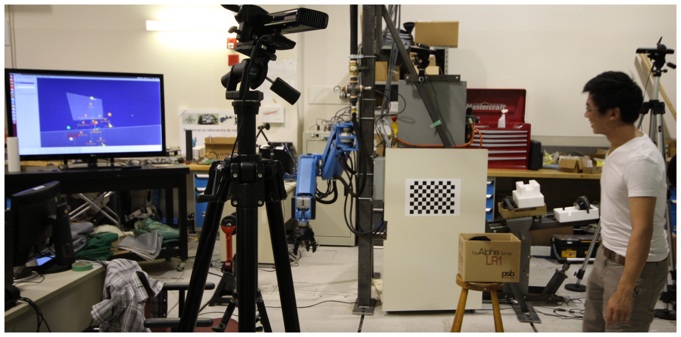

Figure 8. Renderer displays the estimated skeleton to the worker

Some other general task variables were also stored on SADB and accessible, if and when required. As can be seen from the list of variables stated previously, knowledge of the worker's pose is the most important piece of information required to ensure that the system functions efficiently and safely. Both the situational awareness and the human pose tracker thus need to be capable of providing adequately precise values to the robot controller at interactive rates. On an AMD 860K 3.7Ghz quad core processor, we obtained a median performance of 10.2 ± 3.3 Hz and a median latency of 526.7 ± 142.5 ms (Figure 6). The tracking system writes the worker's pose to SADB after every estimate. The pose is stored as a collection of 8 blackboard objects representing the different components of the pose. Approximately 4.61 ms are required to transfer a pose to the blackboard over a gigabit network. The same amount of time (4.70 ms) is required to read a pose back from the blackboard.

For the human workers to feel safe in any collaborative manufacturing environment, the control system needs to provide sufficient feedback to reassure workers that the system functions properly. While providing the numerical value of the tracked pose may be helpful for development and debugging purposes, it may not be adequate to earn trust from the users. For this reason, the tracking system provides a visual renderer capable of depicting how the system perceives the world (Figures 7 and 8). This also allows the workers to re-initialize the tracking as needed and learn the limitations of the system such as how the skeleton estimate is affected by the contact problem.

A. *Discussion*

We observed that failures of the system, as reflected by errors in the skeleton fitting, are largely a result of unresolved contacts. In minor cases, where the fit incorrectly delineates the contacting limb from an object or background, the tracker is able to recover and re-establish correspondence in subsequent views. Although less frequent, there are cases where figure/ground separation can be lost entirely, e.g. multiple objects being tracked in a cluttered environment. The workaround here is for the worker to re-adopt the unambiguous T-pose and re-initialize the tracker. In future work, we hope to address the problem of arbitrary pose initialization.

In our simulated assembly line environment, we have used a 4-Kinect setup to minimize occlusions and maximize worker sampling. Although we believe this is a feasible approach for actual assembly lines, one primary concern is that the size and configuration of the workspace are constrained by the limited sensing range of the cameras and the orthogonal configuration of the sensor network. To alleviate such constraints, in future work, we hope to place additional Kinects at the periphery, overhead or inside the workspace, and it will be interesting to evaluate how the performance of the system scales with increasing cameras.

V. Conclusion

The framework presented here is sufficiently general to accomodate a fairly broad range of systems involving sensors, decision making and actuators. The SADB architecture enables arbitrary dataflow among components with a minimum of contextual information needed to establish links. Sensor networks can easily be established and processes that act upon this data can subsequently build and maintain data structures from which other actors can make decisions. Implementation is platform and language agnostic, and can easily fit into popular substrates such as ROS. Performance, as exemplified by the case study, is quite good using standard commodity hardware, and is scalable up to the speed and latency limits imposed by the underlying network hardware. What we did not fully appreciate at first is that a trust relationship between person and machine must be established for a collaboration to work. The ability to query system states and visualize structures in real-time turns out to be essential in convincing the human co-worker that the robot assistant is sufficiently aware to be trusted.

Acknowledgment

We acknowledge REPARTI, NSERC, GM and NVIDIA for their financial and hardware support.

References

[1] K. N. Mckay, “Computers in industry: Diebold’s view from mid-century,” *International Journal of Computer Integrated Manufacturing*, vol. 13, no. 5, pp. 467–471, 2000.

[2] E. Helms, R. D. Schraft, and M. Hagele, “rob@ work: Robot assistant in industrial environments,” in *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*. IEEE, 2002, pp. 399–404.

[3] J. Krüger, T. Lien, and A. Verl, “Cooperation of human and machines in assembly lines,” *CIRP Annals-Manufacturing Technology*, vol. 58, no. 2, pp. 628–646, 2009.

[4] M. A. Goodrich and A. C. Schultz, “Human-robot interaction: A survey,” *Found. Trends Hum.-Comput. Interact.*, vol. 1, no. 3, pp. 203–275, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1561/1100000005

[5] J. T. C. Tan, F. Duan, Y. Zhang, K. Watanabe, R. Kato, and T. Arai, “Human-robot collaboration in cellular manufacturing: design and development,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 29–34.

[6] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald *et al.*, “The kuka-dlr lightweight robot arm-a new reference platform for robotics research and manufacturing,” in *Robotics (ISR), 2010 41st international symposium on and 2010 6th German conference on robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[7] A. Bicchi, M. A. Peshkin, and J. E. Colgate, “Safety for physical human–robot interaction,” in *Springer handbook of robotics*. Springer, 2008, pp. 1335–1348.

[8] J. Fryman and B. Matthias, “Safety of industrial robots: From conventional to collaborative applications,” in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE, 2012, pp. 1–5.

[9] L. Erman, F. Hayes-Roth, V. Lesser, and D. Reddy, “The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty,” *ACM Computing Surveys (CSUR)*, vol. 12, no. 2, pp. 213–253, 1980.

[10] D. Corkill, K. Gallagher, and P. Johnson, “Achieving flexibility, efficiency and generality in blackboard architectures,” in *In Proc. 6th Nat. Conference on Art. Intelligence (AAAI)*, 1987, pp. 18–23.

[11] H. P. Nii, *Blackboard systems*. Knowledge Systems Laboratory, Depts. of Medical and Computer Science, Stanford University, 1986.

[12] M. Widenius, “Mariadb,” http://mariadb.org/, 2009.

[13] Oracle Corporation, “Mysql,” http://www.mysql.com/, 1995.

[14] Apache Software Foundation, “Couchdb,” http://couchdb.apache.org/, 2005.

[15] S. Sanfilippo and P. Noordhuis, “Redis,” http://redis.io, 2009.

[16] “Mongodb,” http://www.mongodb.org/, 2009.

[17] R. De Chiara, U. Erra, and V. Scarano, “Vennfs: A venn-diagram file manager,” in *Proceedings of the Seventh International Conference on Information Visualization*. IEEE, 2003, pp. 120–125.

[18] A. Phan and F. P. Ferrie, “3d human posture estimation using multiple rgb-d cameras,” in *15th IAPR International Conference on Machine Vision Applications*, 2015, in press.

[19] E. Trucco and A. Verri, *Introductory techniques for 3-D computer vision*. Prentice Hall Englewood Cliffs, 1998, vol. 201.

[20] Opencv camera calibration and 3d reconstruction. Retrieved 2014-04-19. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[21] K. Greff, A. Brandão, S. Krauß, D. Stricker, and E. Clua, “A comparison between background subtraction algorithms using a consumer depth camera.” in *VISAPP (1)*, 2012, pp. 431–436.

[22] Pcl euclidean cluster extraction. Retrieved 2014-05-20. [Online]. Available: http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php

[23] Pcl convexhull class template reference. Retrieved 2014-05-20. [Online]. Available: http://docs.pointclouds.org/1.7.0/classpcl_1_1_convex_hull.html

[24] C. Plagemann, V. Ganapathi, D. Koller, and S. Thrun, “Real-time identification and localization of body parts from depth images,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3108–3113.

[25] L. A. Schwarz, A. Mkhitaryan, D. Mateus, and N. Navab, “Human skeleton tracking from depth data using geodesic distances and optical flow,” *Image and Vision Computing*, vol. 30, no. 3, pp. 217–226, 2012.