A Risk-directed Approach Towards Enhanced Human-Robot Dialog

Junaed Sattar and James J. Little

Abstract—We present an approach for detecting potentially dangerous commands in human-robot dialog, where a robotic system evaluates risk to ask input-specific, directed questions to ensure safe task execution. The goal is to reduce risk, both to the robot and the environment, by asking context-appropriate questions. Given an input program, (i.e., a sequence of commands) the system computes a utility value by evaluating a set of likely alternative programs along with their uncertainty and risk, and then chooses the program with the maximum utility. The chosen program is then evaluated for risk, and whether confirmation from the human partner is necessary. If deemed as such, a process called token-risk grounding identifies the high-risk commands in the programs. We evaluate our system in two simulated robot tasks, and also on-board the Willow Garage PR2 and TurtleBot robots in an indoor task setting. In both sets of evaluations, the results show that by asking relevant, context-sensitive questions, the system provides a better estimate of the likely causes of risk. In addition to ensuring task safety, this results in an overall reduction in robot reprogramming time.

I. INTRODUCTION

Task safety is an essential requirement in robot operations, particularly important in scenarios where robots and humans exist in shared spaces. Risk in the context of robotic task safety may be defined as the cumulative cost of executing a given plan, taken into consideration the working environment and also the robot itself. Ambiguity in communication can also contribute to elevated risk, which need to be eliminated, or at least maximally reduced if complete elimination is not achievable under the circumstances. This reduction of uncertainty and thus risk is an important factor in human-robot interaction (particularly when robots and humans are in close proximity of each other, see Fig. 1). In this work, we address the problem of execution-under-risk in human-robot dialog, using a structured language as the communication medium. This language, called RoboChat [1][2] has been previously used to program and communicate with an amphibious legged robot in both terrestrial and underwater environments. RoboChat enables a human partner to program a robot using an arbitrary communication medium, which may include but would not be limited to visual gestures, audio, and also by using more “traditional” input methods (such as mice and keyboards, or touch interfaces).

Our previous work has looked at quantitatively modeling human-robot dialog, taking into consideration uncertainty in input and corresponding risk (conversely, safety) [3]. In that work, after the human operator programs a robot to perform a set of tasks, the robot assesses risk in command execution and uncertainty in the input, and uses a Decision Function to compute a utility value. Based on the utility value, the robot decides whether or not to confirm this task. The system uses a set of likely programs, derived from the input using a hidden Markov model, to assess execution risk. While the risk assessment helps to prevent potentially unsafe operations from occurring without a confirmation, it does not provide to the human an estimate of risk of the individual elements of the input program (i.e., tokens of the language). Consequently, if a program is evaluated as high-risk, the user would have to confirm (or decline) the program without a sense of the true reason as to why it was evaluated as such. In the worst case, the user may need to re-enter the program in its entirety with corrections in case there is a mistake. For long sequence of instructions, this approach is not user-friendly, tedious, and also increases the chance of erroneous inputs.

This work addresses this specific problem, by not only confirming a risky program, but also by asking specific questions about the parts of the input that contribute to its high-risk (conversely, low-utility) evaluation. The contributions in this paper are the following

1) an algorithm to identify high-risk elements in the input, through a process called token-risk grounding;

2) ensure user intent through multi-stage clarification queries for these specific tokens, and if required, for the final statement as a whole;

that work, after the human operator programs a robot to perform a set of tasks, the robot assesses risk in command execution and uncertainty in the input, and uses a Decision Function to compute a utility value. Based on the utility value, the robot decides whether or not to confirm this task. The system uses a set of likely programs, derived from the input using a hidden Markov model, to assess execution risk. While the risk assessment helps to prevent potentially unsafe operations from occurring without a confirmation, it does not provide to the human an estimate of risk of the individual elements of the input program (i.e., tokens of the language). Consequently, if a program is evaluated as high-risk, the user would have to confirm (or decline) the program without a sense of the true reason as to why it was evaluated as such. In the worst case, the user may need to re-enter the program in its entirety with corrections in case there is a mistake. For long sequence of instructions, this approach is not user-friendly, tedious, and also increases the chance of erroneous inputs.

This work addresses this specific problem, by not only confirming a risky program, but also by asking specific questions about the parts of the input that contribute to its high-risk (conversely, low-utility) evaluation. The contributions in this paper are the following

1) an algorithm to identify high-risk elements in the input, through a process called token-risk grounding;

2) ensure user intent through multi-stage clarification queries for these specific tokens, and if required, for the final statement as a whole;

that work, after the human operator programs a robot to perform a set of tasks, the robot assesses risk in command execution and uncertainty in the input, and uses a Decision Function to compute a utility value. Based on the utility value, the robot decides whether or not to confirm this task. The system uses a set of likely programs, derived from the input using a hidden Markov model, to assess execution risk. While the risk assessment helps to prevent potentially unsafe operations from occurring without a confirmation, it does not provide to the human an estimate of risk of the individual elements of the input program (i.e., tokens of the language). Consequently, if a program is evaluated as high-risk, the user would have to confirm (or decline) the program without a sense of the true reason as to why it was evaluated as such. In the worst case, the user may need to re-enter the program in its entirety with corrections in case there is a mistake. For long sequence of instructions, this approach is not user-friendly, tedious, and also increases the chance of erroneous inputs.

This work addresses this specific problem, by not only confirming a risky program, but also by asking specific questions about the parts of the input that contribute to its high-risk (conversely, low-utility) evaluation. The contributions in this paper are the following

1) an algorithm to identify high-risk elements in the input, through a process called token-risk grounding;

2) ensure user intent through multi-stage clarification queries for these specific tokens, and if required, for the final statement as a whole;
3) ensure safe task execution, and minimize reprogramming time by specific, targeted queries.

For each token in the program, we use a metric to evaluate the probability of that token requiring clarification. While we do not perform strict grounding of tokens (to objects or actions in the physical world, for example), we use the robot’s local (i.e., instantaneous) and global (i.e., environmental) sensor data to calculate the risk contribution of a token within a program, using a set of *risk assessors*. The evaluation takes place in batch mode, and no clarifications are asked until the end of the input is reached. Once all clarifications are received from the human, the command is reevaluated, and if needed, a confirmation request is generated.

II. RELATED WORK

The research presented in this paper extends the authors’ prior work on human-robot dialog in the presence of uncertainty and risk. The dialog mechanism relies on having an explicit means of communication between a robot and a human partner. A rich, varied ensemble of communication modalities have been used in human-machine interfaces, both in robotics and human-computer interaction. Also, we apply a Markovian dialog model to infer intentions from dialog and to measure uncertainty. We present previous work in each of these disparate domains in this section.

Sattar et al. [1] investigated the use of visual communications, by using a gesture-based language using fiducials [4], and also looked at person-following by an underwater vehicle by tracking the swimming motion of a human “dive-buddy” [5]. By combining these two approaches, a vision-based human-robot interaction architecture has been developed [6]. In that work, while “closed-loop” robot control is achieved, an explicit dialog-based interface between the human and the robot did not exist. This paper is motivated partially from that work. Visual communication has also been used by several researchers for communication between a network of heterogeneous robotic systems, for example by Dunbabin et al. [7].

Waldherr [8] et al. demonstrated an explicit communication paradigm in which hand gestures are used to interact with a robot and lead it through an environment. Tsotsos et al. [9] considered a 3D gestural interface for non-expert users, in particular disabled children, based on a combination of stereo vision and keyboard-like input. As an example of implicit communication, Rybski and Voyle [10] developed a system whereby a robot could observe a human performing a task and learn about the environment.

Gesture-based robot control has been considered extensively in Human-Robot Interaction (HRI). This includes explicit as well as implicit communication frameworks between human operators and robotics systems. Several authors have considered specialized gestural behaviors [11] or strokes on a touch screen to control basic robot navigation. Skubic et al. have examined the combination of several types of human interface components, with special emphasis on speech, to express spatial relationships and spatial navigation tasks [12].

While our current work looks at interaction under uncertainty in any input modality, researchers have investigated uncertainty modeling in human-robot communication with specific input methods. For example, Pateras et al. applied fuzzy logic to reduce uncertainty to reduce high-level task descriptions into robot sensor-specific commands in a spoken-dialog HRI model [13]. Montemerlo et al. have investigated risk functions for safer navigation and environmental sampling for the Nursebot robotic nurse in the care of the elderly [14]. Bayesian risk estimates and active learning in POMDP formulations in a limited-interaction dialog model [15] and spoken language interaction models [16] have also been investigated in the past. Researchers have also applied planning cost models for efficient human-robot interaction tasks [17] [18]. Scheutz et al. [19] have looked at enhanced natural language dialogs for human-communication. In [20], the authors use information from natural language dialogs to update the robot task planner in real-time. Also, Chernova et al. [21] apply crowdsourcing to address real-world human-robot dialog through online multiplayer games.

In recent work, Tellex et al. [22] looked at asking directed questions to reduce uncertainty in human-robot dialogs, specifically for natural language communication. The authors use a graph-theoretical approach to ground tokens in the language to elements in the world to detect points of uncertainty. While somewhat similar to the problem we are addressing, our approach differs from their work in the cues being used to direct confirmations. In particular, their work looks at questions asked by a robot to reduce input language ambiguity, where as in our case, we use an evaluation of task risk (i.e., cost) to direct specific questions to the user. Similarly in [23], the authors address the issue of assessing and informing non-expert users about the feasibility of the given natural language command. The particular concern in our work is to prevent the robot from executing potentially risky tasks without confirmation, yet still make it possible to do so if the user so desires. Specifically, we aim to prevent risky task execution without confirmation, whether the commanded task was intentional or not. While we do not use natural language directives in this work, such commands can be accommodated as long as a mechanism to produce possible alternatives along with their likelihoods exists (e.g., using the Google voice input API [24]).

III. TECHNICAL APPROACH

The core of this work involves a dialog between a human and a robot, towards executing a set of tasks. The goal is to enable the robot to ask targeted questions, in order to reduce uncertainty and obtain a true measure of risk involved in the commanded tasks, and eventually asking for confirmation if deemed necessary. To achieve this, we rely on generating a set of task execution plans from the most-likely programs chosen by the robot, and assess the risk of these plans by using a set of domain-specific assessors. Once the risk-uncertainty values are obtained, we use a utility measure to choose the most-likely of the programs, and verify if the task
is high-risk. If so, the tokens in the input are analyzed by a risk-analysis component, looking at the world representation (in the form of global and local sensory information) available to the robot, and “grounding” tokens in the command to elements in this representation. This grounding step identifies potential high-risk commands by applying the assessors on the plan, pushing the ‘suspect’ tokens into a clarification queue. At the end of the risk analysis step, the robot asks the human operator for clarification for all tokens in the queue (i.e., in the form of confirmation or re-entry). When all tokens from the queue are clarified by the user, the overall risk is calculated for the (potentially) modified program, and passed to the decision function. At this stage, based on the output of the decision function, the program is either carried out immediately, or a final confirmation is asked from the user. Note that we make an important distinction between ‘safety’ and ‘correctness’ in this approach. While the purpose of the proposed algorithm is to ensure risky tasks are executed only after a confirmation, it will in no way ensure that the “correct” program (as intended by the user) has been input. In other words, as long as the program is evaluated as safe, the system will allow it to execute. Thus it is quite possible for a wrong but safe program to be executed, without a feedback from the system. The following sections take a detailed look at the various steps of this algorithm.

A. Language structure

The robot language used in this work is based on the RoboChat syntax. For the sake of brevity, we assume that a mechanism exists to only allow syntactically valid and semantically consistent commands to be sent to the robot. Details of the language structure can be found in [3], but for the sake of completeness a summary is given here:

1) A program, $G$, is described as consisting of a finite number of tokens, $g_i$.
2) A program must start with a command token, $g_c$, and is optionally followed by a parameter, $g_p$.
3) We assume each token is independently uttered by the operator, and thus the probability of a sequence of tokens chained together to form a “compound statement” simply becomes:

$$p(S) = p(g_{c1}, [g_{p1}], g_{c2}, \ldots, g_{cn}, [g_{pn}]) = \prod_{i=1}^{n} p(g_i)$$  \hspace{1cm} (1)$$

where $p(g_i)$ is the probability of the token $g_i$ being used (i.e., uttered) by the human operator, for both commands or parameters. Thus, if a table of values for all $p(g_i)$ is available, it is straightforward to compute the probability of a command (i.e., a sequence of tokens).
4) Each program $G_i$ has a likelihood of occurrence $l_i$ and a cost $c_i$ associated with it. It is worth noting however, given the input language, the set of all
possible programs will be reduced, as the inconsistent ones, both syntactically and semantically, are going to be expunged.

B. Risk assessors

Once the system has resolved the uncertainty and chosen a set of likely programs to choose from, the next step is to assess the risk in each of these programs. Risk, as defined in the context of this work, refers to the cost of executing a given command. A high-risk command is evaluated as unsafe by the algorithm, and vice-versa. Risk is evaluated both in terms of risk to the environment and the robot itself, and also the inherent costs associated with operating a robot, such as wear-and-tear, battery consumption, computational load etc. Given the mostly-likely input, the algorithm generates a plan to carry out the given program. This plan is then simulated and the cost of execution is evaluated by a set of assessors, simultaneously as the task is simulated. After executing each individual token in the input, these assessors examine the current state of the robot and compute a conservative value of risk (i.e., not less than the true risk). The final program cost is a sum of all the assessors’ outputs over the duration of the simulated program.

C. Token-Risk grounding

Once a likely program has been chosen and is evaluated as high-risk, the system attempts to analyze the program to identify possible tokens that contributed to the overall risk of the system. Owing to the inherent structure of RoboChat, the components can contribute to the risk in one of two possible ways, given a (semantically and syntactically) valid command. In the first, a command token, irrespective of the parameter for the command, would cause the plan to take a potentially unsafe or costly path, and thus increase the risk factor. Secondly, the command may not be evaluated as costly on its own (i.e., independent of a parameter), but a misinterpretation of the parameter (caused either by a mistake by the programmer or uncertainty in the communication medium) would raise the risk factor significantly.

To illustrate the point, consider the following two program segments given to an unmanned aerial vehicle in RoboChat syntax:

1) TAKEOFF, GEARUP, ALTITUDE 100, , RECORD_IMAGE_DATA, ALTITUDE 2, ...

2) GEARUP, TAKEOFF, ALTITUDE 100, RECORD_IMAGE_DATA, ALTITUDE 200 , ...

In the first program, the last ALTITUDE command is given a parameter of 2 meters. The system would evaluate the parameter value as too low an altitude for the aircraft to operate safely, and thus place a high-cost value at that token. On the other hand, in the second program, the plane is commanded to lift up its landing gear before it has taken off. Though no parameters are required, since lifting the landing gear before take-off would cause significant damage to the aircraft, the risk assessors would consider this command unsafe.

Algorithm 1 Token-risk grounding and dialog clarification.

1: λ ← Input Command
2: if λ ≠ φ then
3:   for all t_j ∈ λ do
4:     Risk_j ← ComputeTokenRisk(t_j)
5:   end for
6:   Risk_λ ← Assess(λ)
7:   Risk_mean ← ComputeMeanRisk(Risk_λ)
8:   for all Tokens, t_j ∈ λ do
9:     repeat
10:        Risk_temp ← Risk_j
11:        Risk_j ← 0
12:        Risk_new ← Assess(λ)
13:        if |Risk_new − Risk_λ| ≥ Risk_mean then
14:           Push(t_j, Q_C)
15:       end if
16:       Risk_j ← Risk_temp
17:     until Risk Assessment Complete
18:   end for
19:   if Q_C ≠ φ then
20:     while Q_C ≠ φ do
21:        T ← Pop(Q_C)
22:        Clarify(T)
23:     end while
24:   end if
25: end if
both on-board and off-board experiments, we use a RoboChat vocabulary consisting of motion commands, surveillance (i.e., recording photos and videos) and manipulation commands (see Tables I and II for a sample of language tokens used), with the execute command marking the end of input. In the off-board experiments, input modality is limited to mouse input, whereas the robot trials are performed using a ROS [25] module as a back-end to the mouse input system, to communicate commands over an IP network. The ROS navigation stack is used by the robots to localize and navigate in the environment.

To calculate uncertainty in input for each of the simulated robots, a Hidden Markov Model is trained with estimated parameters for the given vocabulary. To estimate task costs, we use two custom simulation models for each type of robot. For the underwater vehicle, we use a set of assessors that takes into account the operating contexts of an autonomous underwater vehicle, and use the simulator used in [3]. The simulator has been designed to take into account the robot’s velocity, maneuverability and propulsion characteristics to accurately and realistically simulate trajectories taken by the robot. For the terrestrial robots, the simulator design is based upon the ROS simulation packages for the PR2 and the TurtleBot robots.

Since the goal of the off-board evaluations are to quantify the performance of the proposed algorithm in identifying potentially risky parts of a given command, we do not require the simulated robots to execute any tasks, but only perform token-risk grounding and ask clarifications if judged necessary. Results from both sets of experiments are presented in the following sections, preceded by a brief description of the input assessors and the actual tasks that were given.

A. Assessors

For both simulated and on-board robot trials for the indoor environment, we use the following assessors to evaluate risk:

1) Total distance: The operating cost and risk factors both increase with total distance traveled by the robot.
2) Average Distance: While the farthest and total distance metrics consider extremes in range and travel, respectively, the average distance looks at the distance of the robot (from start location) where most of the task execution time is spent.
3) Execution Time: An extremely long execution time also carries the overhead of elevated operational and external risk.
4) Preconditions: Certain functions need to have parameters initialized before they can be executed, and also depend on other functions having already executed.
5) Navigability: If the robot cannot safely navigate to a given location, the chance of damage to the robot from collision increases, as does the probability of task failure.

Also, for the simulated underwater robot, in place of the Navigability assessor, we use the following two additional assessors:

1) Farthest distance: The farther the robot goes from the initial position, the higher the chance of losing the robot.
2) Depth: The deeper the robot dives under the surface, the higher the chances are of exceeding maximum operational depth, thus irreparably damaging the vehicle. Also, certain operations, such as getting a GPS fix or bearing is not possible at depth, and thus would be flagged if attempted.

B. Simulated tasks

The programs used for the simulated underwater robot are shown in Tab. I, along with the decision of confirmation. The numerical arguments are either distance measures (in meters, for the motion commands) or time (in seconds, for the non-motion commands, such as in PICTURE). Table. II shows the programs sent to the indoor robot, operating in the annotated map shown in Fig. 3.

C. Clarification Queries

Our focus on this paper is the identification of risky commands and specific high-risk components within them. While the output generated by the token-risk grounding algorithm can be used in an arbitrary fashion to provide detailed feedback and further planning, for the experiments,

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequence</th>
<th>Risky?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FORWARD, 30, PICTURE, LEFT, 30, PICTURE, SURFACE, GPSFIX, GPSBEARING, EXECUTE</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>FORWARD, 180, LEFT, 20, FORWARD, 180, MOVIE, 300, RIGHT, 15, GPSFIX, SURFACE, STOP, EXECUTE</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>LEFT, 30, RIGHT, 10, MOVIE, 120, FOLLOW, 60, SURFACE, GPSFIX, EXECUTE</td>
<td>Yes</td>
</tr>
</tbody>
</table>

TABLE I: Commands sent to the simulated underwater robot.
Table II: Commands sent to the simulated terrestrial robot.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequence</th>
<th>Risky?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>GOTO, JAYS_DESK, PICKUP, CUP, GOTO, KITCHEN, PUTDOWN, CUP, GOTO, FIRE_ESCAPE, TAKE_PICTURE, STOP, EXECUTE</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>GOTO, JAYS_DESK, PICKUP, BOOK, GOTO, DANS_ROOM, PUTDOWN, BOOK, GOTO, JAYS_DESK, EXECUTE</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>GOTO, COFFEEMAKER, PICKUP, MUG, GOTO, DANS_ROOM, PUTDOWN, MUG, PICKUP, DVD, GOTO, TV, PUTDOWN, DVD, GOTO, BATHROOM, EXECUTE</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table III: Contents of the clarification queue, and assessors that evaluated the tokens as high-risk.

<table>
<thead>
<tr>
<th>ID</th>
<th>Assessor Flags</th>
<th>Clarification Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>FARTHEST_DISTANCE, TOTAL_DISTANCE, DEPTH, PRECONDITION</td>
<td>TBA, TBA, GPSFIX</td>
</tr>
<tr>
<td>3</td>
<td>PRECONDITION</td>
<td>FOLLOW</td>
</tr>
<tr>
<td>5</td>
<td>NAVIGABILITY</td>
<td>DANS_ROOM</td>
</tr>
<tr>
<td>6</td>
<td>TOTAL_DISTANCE, NAVIGABILITY</td>
<td>COFFEEMAKER, DANS_ROOM, BATHROOM</td>
</tr>
</tbody>
</table>

we limit the feedback query to take the following form:

\[
\text{CLARIFICATION\_QUERY} \leftarrow \text{“Did you mean ”} + \text{HIGH\_RISK\_TOKEN} + \text{“?”} \quad (2)
\]

where the + symbol denotes string concatenation. The users only have to either confirm the identified high-risk token, or to change it, edit that one token only.

D. Results & Discussion

In the simulated tasks from Tables I and II, the algorithm correctly evaluated commands 2, 3, 5 and 6 as ‘risky’. Commands 1 and 4 were evaluated as ‘safe’, and accordingly, no clarifications were requested for these programs. The contents of the clarification queue for the ‘risky’ programs are shown in Tab. III, along with the assessors that contributed to the high-risk evaluation. The user was given an option to change the tokens in commands in-place, if they so desired.

In command 2, the distances given to the simulated underwater robot was large, and would cause the robot to travel a great distance from the initial position, thereby increasing the chances of losing the robot. Thus the system requested clarification on the distances. Furthermore, the GPSFIX command was performed before the SURFACE command, causing the precondition assessor to put a high cost on this token, and placing it in the clarification queue.

In command 3, the FOLLOW instruction is given to visually track and follow a target. But since the tracker has not been initialized (in this case, with a TUNETRACKER command) the precondition assessor flags the instruction as high-risk.

In command 5, the location referred to as “DANS_ROOM” is not reachable by the robot, as no paths exist in the map (seen in Fig. 3). A failure of the navigation task would possibly occur and thus the target location is marked for clarification.

In command 6, a navigability issue exists, similar to command 5, to reach the “COFFEEMAKER” location. Moreover, the overall distance traveled by the robot to navigate between the location is assessed to be too large, and thus a clarification is requested for the final location, “BATHROOM”.

E. Timing

By asking for targeted feedback, it was possible to achieve gains in programming times. Specifically, there were three possible outcomes after a program was given:

1) No feedback required: in such cases, the overall programming time was unchanged.
2) Feedback required, tokens unchanged: in such cases, the overall programming time increased because of the additional clarification queries asked by the system, plus the time taken by the users to answer those queries.
3) Feedback required, one or more token changed: in such cases, we achieved a reduction in programming time. In particular, for long programs (i.e., many tokens), the cost of re-programming without targeted queries is higher, as the user needs to input the entire sequence (plus the changes) again.

To measure the changes in timing, we performed 20 trials each of the three different feedback cases discussed above, with and without token-risk grounding, resulting in 120 total experiments. Figure 4 shows the percentage changes in programming time obtained from these trials as compared to a system without token-risk grounding (i.e., the baseline case obtained from exactly the same tasks with no token-risk grounding performed, but with overall risk computed). The results show approximately 40 per cent saving in programming time from a system without token-risk grounding when the user is using the feedback process to make changes to the original input program.

F. Robot trials

Two separate robot trials were performed to qualitatively assess the performance of the algorithm on-board a robotic system. Both robots, one a PR2 and the other a TurtleBot, were given a set the three commands comprised of navigation and surveillance tasks (see Tab. II) in the same environment (see Fig. 3), and thus the map data was shared between the robots. Note that the map data was obtained prior to the experiments by using the ROS gmapping method using the PR2 robot. The manipulation tasks were manually performed on the PR2. Also, as the TurtleBot is not equipped with a manipulator, no manipulation tasks were performed on that robot. Input was given to the robots on a desktop PC, using a mouse gestures-driven interface. Through a ROS module developed using ROS messaging stack. A gesture- or speech-based interface would be equally applicable, but since the focus of our work was on the problem of asking targeted questions, we chose an arbitrary input modality. The only
difference in using another modality would lie in uncertainty evaluation.

Clarification of the tokens were asked of the user over the same module, on the a desktop PC. The user had an option of either repeating the same command/parameter, or entering a new token in place of the previous one. After the clarification stage was complete, the robot either asked a final task confirmation (in case of a high-risk task) or executed the given command. The trial performances were similar to the simulation runs, with the difference that the robots actually carried out the task once they were confirmed by the operator.

V. CONCLUSIONS

We present an algorithm for a robotic system to ask task- and context-specific questions to a human operator to reduce uncertainty and improve safety in task execution. The system assesses uncertainty and risk in human-robot dialog, and finds ‘risky’ language tokens from the input command by grounding evaluated with risk with language tokens. Specifically, we associate tokens in the input command with actions in the world representation of the robot to detect tokens that cause a significant increase in overall risk. The system has been evaluated with a set of simulated tasks for an indoor robot, and also evaluated on-board the PR2 and Turtlebot platforms.

As an immediate next step, we are planning large-scale human interface trials on real and simulated robots (and environments), and measure the impact of this approach in terms of time savings, ease of communication, and feedback relevance. Future work is planned to enhance the grounding of the task set to both actions and objects in the real-world representation of a robot. We intend to investigate the effects of modeling risk as a distribution, dynamically updating risk assessment using on-the-fly sensing, and thus updating the token-risk grounding mid-task. Also of interest is a shared estimation of risk-grounding, where a network of mobile robots and static sensors, including portable devices (e.g., smartphones) build and share a representation of the world, and assist in distributed and coordinated human-machine interaction. Particularly in the domain of assistive robotics (e.g., autonomous wheelchairs for the mobility impaired), a shared HRI network with a strong grounding of risk to objects and actions would help to establish a safe and secure domain of operations, yet maintain a natural, intuitive modality in the human-machine interface.

REFERENCES


Fig. 4: Relative change in programming time with token-risk grounding. The zero-percent line is the baseline case without token-risk grounding.


