Self-Repair and Self-Extension by Tightening Screws based on Precise Calculation of Screw Pose of Self-Body with CAD Data and Graph Search with Regrasping a Driver

Takayuki Murooka, Kei Okada, and Masayuki Inaba

Abstract—In this paper, we propose methods for tightening screws of self-body using a driver, which enable self-repair and self-extension. There are two difficulties for tightening screws of self-body. First, the precise calculation of the screw pose is needed. When calculation with visual images using a camera, the observation error is so high. The merit of the robot is that the robot has CAD data of self-body. There we calculate the precise screw pose with self CAD data. Second, because of the small closed links when tightening screws of self-body, that the robot cannot move the driver for rotating around the screw sometimes happens because inverse kinematics cannot be solved. To solve this problem, we propose a method of tightening motion generation with regrasping a driver if inverse kinematics cannot be solved. With these methods, humanoid robots PR2 and HIRO realized self-repair and self-extension by tightening screws of self-body.

I. INTRODUCTION

Humanoid robots are expected to work continuously in the home environment or disaster sites. However the hardware of the robot sometimes breaks, and needs to be repaired. So it is essential for humanoid robots to obtain self-repair methods. We human can become aware of our own abnormalities with visual information, plan repairing, and execute that plan using appropriate tools if needed. Like this, we want humanoid robots to repair self-body by oneself. We come up with loose screws of the robot which occurs sometimes and needs to be repaired. In this study, we focus on the robot tightening loose screws by oneself as self-repair. In addition, the robot can realize self-extension by tightening screws, so we also focus on self-extension.

One of the advantages of robots including humanoid robots is that they have their own software/hardware design information. This time, by using its own CAD data, which describe hardware information, the robot realizes calculating the screw pose more precisely than vision-based calculation. As a format of CAD data, STEP file is widely used as an intermediate file, so we use this format. We extract each part from STEP file using FreeCAD [1].

In the case of tightening the screw of an external object rather than the robot itself, the robot can move relatively to the object, whereas in the case of tightening the screw of self-body, the solvability of inverse kinematics is considerably limited because of the small closed links when tightening the screw of self-body. So, in this research, when inverse kinematics cannot be solved, it is made to be able to solve inverse kinematics by changing the grasping pose without changing the pose of the tool. This we call regrasping. Combining these methods, we aim that robots tightening screws of self-body without any help.

II. RELATED WORKS AND PROPOSED SYSTEM

The topics about self-repair, self-heal or self-fix have been widely researched. There are modular robots which can reconfigure by oneself [2], [3], [4]. However there are limits of strength, precision, and mechanical and electrical stability with modular robots. Other solutions of the self-repair system like mechanism improvement to the screw itself [5], [6] or SH (self-healing) soft materials [7], [8] were developed.

However it is hard to incorporate these mechanisms or materials into the existing robots because it needs much cost and efforts to remodel. So it is better for robots to repair self-body using hands like human, especially if the robot is a humanoid robot. About the research of tightening screws with hands, the system which enables tightening screw with a robot arm [9] or with human-robot interaction [10] were developed, but in this case, a driver is fixed with the robot.

The problem of the task of tightening screws is that the precise information about the screw pose is needed. In the task which needs precise information like tightening screws or peg-in-hole, the method using a force-torque sensor was proposed [11], [12], or vision-based method was also proposed [13], [14]. However calculating the screw pose directly from a force-torque sensor is of course impossible, so the planner to insert the screw or the peg is needed, and the vision-based system depends on the performance of the camera of the robot, and that camera is not always precise enough to calculate the precise screw pose.
A. Overview of Tightening Screws System

We show the overview of the tightening screw system in Fig. 2. The robot thinks about taking up tightening screws of self-body with some triggers. For example, the robot checks whether there is any abnormality of self-body as a daily check, or perceives the loose screws by the pose of links of the robot which is out of ideal pose or by other people’s indication from the abnormality. Then from the information about that trigger, the robot calculates the position of the screw to tighten roughly (A). With self CAD data, the robot calculates the precise screw pose to tighten (B). After calculating the precise screw pose, the robot generates the motion of tightening, with regrasping a driver if it is needed to tighten (C).

III. Perception of Loose Screw

In this section, we will show two examples of perceiving the loose screws. One is that the robot perceives the loose screw by oneself and another is that the robot perceives the loose screw by other people’s indication with interactive communication.

A. Perception by Oneself

The robot can calculate the pose of each link of the robot by calculating joint angles with joint encoders. One example of the bad result caused by loose screws, which is much visible, is that the pose of a link is out of ideal pose like Fig. 3. In detail, applying a mask image of self-body, whose black region means the robot and whose white region means others to a RGBD image observed with the RGBD camera, the RGBD image which describes only the robot can be calculated. Then the robot reconstructs point cloud from that RGBD image, then the robot can get the point cloud of self-body. If there are some points out of ideal, the pose of a link is decided as out of ideal. That means there may be the loose screw, so this information is sent to the next step of ‘Screw Pose Calculator’ in Fig. 2.

B. Perception by Other People’s Indication

Of course the robot cannot perceive all loose screws by oneself, so it is important to be taught that there is a loose screw by other people’s indication with interactive communication like Fig. 4. In this study, we can teach the loose screw position to the robot by pointing with a finger and the voice like ‘There is a loose screw’. We implemented calculating people pose with OpenPose [15]. Then the robot perceives that there is a loose screw, and calculates the position of the loose screw roughly with the tip of the finger position of pointing. The robot sends this information to the next step of ‘Screw Pose Calculator’ in Fig. 2.

IV. Calculation of Screw Pose

A. Screw Judgement

We aim to detect the screw which is closest to the designated position which is calculated with Sec. III. At first, we describe an algorithm to judge whether the part which constructs the robot is a screw or not in Alg. 1. Each part in the STEP file is expressed in units of shell, and each shell has information of edges and vertices that constitute the part.
We describe functions in Alg. 1 in the following. \(l_{\text{max}}, l_{\text{min}}\) are shown in Fig. 6.

**Algorithm 1 Judging Whether Shell is Screw or not**

1: function JUDGE WHETHER SCREW (shell)
2:   \(C \leftarrow []\)
3:   for edge in shell.Edges do
4:     if IsCircle(edge) then
5:       \(\text{radius} \leftarrow \text{calcRadiusOfCircle}(\text{edge})\)
6:       \(\text{direction} \leftarrow \text{calcDirectionOfCircle}(\text{edge})\)
7:       \(p_{\text{center}} \leftarrow \text{calcCenterOfCircle}(\text{edge})\)
8:       \(\text{circle info} \leftarrow (\text{radius}, \text{direction}, p_{\text{center}})\)
9:       \(\text{push(circle info, } C)\)
10:   end if
11: end for
12: if \(C.\text{length} < \text{threshold}_{1}\) then
13:   return False
14: end if
15: if \(\text{calcConcentricMaxRatio}(C) < \text{threshold}_{2}\) then
16:   return False
17: end if
18: \(C_{\text{concentric}} \leftarrow \text{calcConcentricCircles}(C)\)
19: \(l_{\text{max}} \leftarrow \text{calcMaxLength}(C_{\text{concentric}})\)
20: \(l_{\text{min}} \leftarrow \text{calcMinLength}(C_{\text{concentric}})\)
21: if \(l_{\text{max}}/l_{\text{min}} < \text{threshold}_{3}\) then
22:   return True
23: else
24:   return False
25: end if
26: end function

- **isCircle(edge)**: This function judges whether the edge is circle or not. Each edge has that information.
- **calcRadiusOfCircle(edge)**: This function calculates the radius of the circular edge.
- **calcDirectionOfCircle(edge)**: This function calculates the vertical direction of the circular edge.
- **calcCenterOfCircle(edge)**: This function calculates the center position of the circular edge.
- **calcConcentricMaxRatio(list)**: This function calculates the max ratio of the circles which are concentric each other in all the circles in list.
- **calcConcentricCircles(list)**: This function calculates the circles which are concentric each other and the ratio of the circles in all the circles is maximum in list.
- **calcMaxLength(list)**: This function calculates the most far distance of the centers of the circular edges whose radius is maximum in all the circles in list.
- **calcMinLength(list)**: This function calculates the most far distance of the centers of the circular edges whose radius is minimum in all the circles in list.

Using this screw judgement algorithm, we extracted screws from three parts: UR10 (Universal Robot 10)\(^1\), the left shoulder of PR2, and the base parts we designed. The results are shown in Fig. 5. All results show that the screw is correctly extracted. Some thresholds in Alg. 1 must be tuned according to the characteristic of screws of the parts, but we can extract screws of Fig. 5 with the same thresholds. At last, the closest screw among screws extracted with Alg. 1 to the designated points of Sec. III, we determine the target screw to tighten.

**B. Calculation of Screw Pose**

The flow of calculation of the screw pose is shown in Fig. 7. Now we aim to find the screw pose with the left shoulder of the robot. \(p_{b}^{a}\) describes the pose of a in the coordinate system of b, and \(bT_{a}\) describes the translation matrix from the coordinate system of b to the coordinate system of a. The robot calculates \(\text{base}T_{\text{link}}\) with each joint angle and the geometric model of the robot, and also calculates \(\text{link}T_{\text{screw}}\) with \(d\) and \(p_{\text{center}}\) of the target screw calculated by Alg. 1. We can get \(p_{\text{base}}^{\text{screw}}\) by calculation of \(\text{base}T_{\text{screw}} = \text{base}T_{\text{link}} \times \text{link}T_{\text{screw}}\).

**Fig. 7. Calculating the target screw pose from the coordinate system of the base link.**

**V. Motion Generation of Tightening Screws with Regraspers**

**A. Inverse Kinematics Considering a Driver and a Screw**

After grasping the driver, inverse kinematics is calculated so that the pose of the tip of the driver and the pose of the screw head will match. Assuming that the pose of the tip of the driver is \(\tau_{\text{driver}}\), and the Jacobian of the robot link related to the movement of the pose of the tip of the driver \((\tau_{\text{driver}})\) is \(J_{\text{driver}}\). Similarly, assuming that the pose of the screw head is \(\tau_{\text{screw}}\), and the Jacobian of the robot link related to the movement of the pose of the screw head \((\tau_{\text{screw}})\) is \(J_{\text{screw}}\). We can solve whole-body inverse kinematics, which combines the robot links of not only a sole arm but also both arms, with a driver and a screw head, by iterative calculation by using \(\Delta \Theta\) as Eq. (1). The result of inverse kinematics is described in Fig. 8

\[
\begin{align*}
\Delta \tau_{\text{driver}} &= J_{\text{driver}} \Delta \Theta_{\text{driver}} \\
\Delta \tau_{\text{screw}} &= J_{\text{screw}} \Delta \Theta_{\text{screw}}
\end{align*}
\]

Suppose \(\Delta \tau_{\text{driver}}, \Delta \tau_{\text{screw}}\) as follows.

\[
\begin{align*}
\Delta \tau_{\text{driver}} &= \tau_{\text{driver}} \Delta \tau_{\text{screw}} \\
\Delta \tau_{\text{screw}} &= \tau_{\text{driver}} \Delta \tau_{\text{screw}}
\end{align*}
\]

\(^1\)https://www.universal-robots.com/products/ur10-robot/
When using the robot which has a force-torque sensor on the robot’s limbs, the robot can judge whether regrasping is feasible or not by Eq. (3), Eq. (4) while executing regrasping because the robot knows \( \mathbf{f}_{\text{hand}} \) and \( m_{\text{hand}} \). However the robot which doesn’t have a force-torque sensor cannot judge feasibility only with Eq. (3), Eq. (4). So we consider constraints for the robot which doesn’t have a force-torque sensor with some approximation in the following.

1) When Translational Regrasping: To convert the constraint of force and moment to the constraint of position and orientation, we formulate below equations. \( \mu_r \) is the translational coefficient of static friction between the robot’s hand and the driver. \( e_s \) describes the unit vector of screw vector. \( \Delta r_{\text{pos}} \) is a two-dimensional unit vector, describing the direction of translational regrasping movement. \( N \) is force of grasping a driver.

\[
\begin{cases}
|\mathbf{f}_{\text{hand}}| = \mu_r N \\
\epsilon_{\text{f}_{\text{hand}}} = \epsilon_{\Delta r_{\text{pos}}} 
\end{cases}
\]

Then we apply strong approximation obtained empirically. When regrasping motion is only translational, we assume that \( m_{\text{hand}} \) equals zero. So we can formulate \( \mathbf{f}_{\text{screw}}, \mathbf{m}_{\text{screw}} \) with the pose of the robot’s hand information by combining Eq. (3), Eq. (4), Eq. (5). \( N \) is assumed to be known.

2) When Rotational Regrasping: To convert the constraint of force and moment to the constraint of position and orientation, we formulate below equation. \( \mu_r \) is the rotational coefficient of static friction. \( \Delta \mathbf{r}_{\text{ori}} \) is a value whose absolute value is 1, describing the direction of rotational regrasping movement. \( \text{sgn}(\text{val}) \) returns sign of val.

\[
\begin{cases}
|\mathbf{m}_{\text{hand}}| = \mu_r N \\
\text{sgn}(m_{\text{hand}}) = \text{sgn}(\Delta \mathbf{r}_{\text{ori}})
\end{cases}
\]

Then we apply strong approximation obtained empirically. When regrasping motion is only rotational, we assume that \( m_{\text{screw}} \) equals zero. So we can formulate \( \mathbf{f}_{\text{screw}}, \mathbf{m}_{\text{screw}} \) with the pose of the robot’s hand information by combining Eq. (3), Eq. (4), Eq. (6). \( N \) is assumed to be known.

To verify the approximate constraint described upper, we conducted experiments of regrasping with PR2 like Fig. 10. The robot which doesn’t have a force-torque sensor did translational regrasping and rotational regrasping which is considered to be feasible according to the approximate constraint, 20 times respectively. The direction of regrasping is random. The movement of translational regrasping is 1cm, and the rotation angle of rotational regrasping is 15 degree. The result of the experiment is shown in Tab.I. Success means that regrasping is realized without the driver moving. It is considered that the approximation that \( m_{\text{screw}} \) equals zero is strict.

This regrasping is not needed if the axis of rotation of the robot’s grasping hand and the axis of rotation of the driver is on the same line. If that matching is difficult because of the less solvability of inverse kinematics of the small closed links, this regrasping is effective.


TABLE I. Result of experiment of regrasping

<table>
<thead>
<tr>
<th></th>
<th>translational regrasp</th>
<th>rotational regrasp</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>fail</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>total</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>success rate</td>
<td>0.9</td>
<td>0.75</td>
</tr>
</tbody>
</table>

We show the examples of motion of regrasping a driver in Fig. 12. The robot succeeded rotating a driver one round with regrasping twice.

![Fig. 12. Tightening a screw with regrasping a driver. The robot does regrasping from ③ to ④.](image)

VI. EXPERIMENTS OF SELF-REPAIR AND SELF-EXTENSION

With the proposed system shown in Fig. 2, we conducted experiments of self-repair and self-extension with a real life-size humanoid robot using a driver designed for human.

A. Self-repair as Daily Check by HIRO

![Fig. 13. HIRO tightened a screw of self-body as a daily check.](image)

The humanoid robot HIRO executes self-repair as a daily check. HIRO calculated a screw pose with self CAD data and moved a proper driver to the screw, then tightened. Shown in Fig. 13, HIRO succeeded in tightening a screw of self-body.

B. Self-repair by Other People’s Indication by PR2

![Fig. 14. PR2 perceived a loose screw by other people’s indication and executed tightening the screw with a proper driver. PR2 executed regrasping from ③ to ④.](image)

The humanoid robot PR2 executes self-repair by other’s indication. A people noticed the loose screw of PR2, and said ‘This screw is loose’. Then PR2 perceived the loose screw, calculated the rough position of the screw and also calculated precise the screw pose with self CAD data. After that, PR2 started tightening the screw with a proper driver. We show the snapshots of the two examples when PR2 tightened a screw of self-body with regrasping in Fig. 14.

C. Motion Generation of Tightening Screw by Graph Search

We aim to generate motion of tightening a screw. Because of the small closed link when tightening screws of self-body, that the robot cannot move the driver to rotate around the screw sometimes happens. That means inverse kinematics of Sec. V-A cannot be solved. To solve this problem, we consider inverse kinematics with the regrasping with some constraints described in Sec. V-B.

First, the rotation of the driver is advanced from the initial rotation angle of the driver \(\phi_{\text{start}}\). If inverse kinematics cannot be solved, the robot searches the next grasping pose with the constraints of Sec. V-B. If inverse kinematics can be solved with the next grasping pose, the rotation of the driver is advanced, and if not, the robot searches the next grasping pose. This search is executed as a depth-first search, considering two kinds of infeasibility: inverse kinematics and regrasping. We show this search in Fig. 11 and the algorithm.

![Fig. 11. Graph search of tightening motion. A vertical axis describes the grasping pose of the driver, and a horizontal axis describes the rotation angle of the driver of tightening motion. Blue nodes are feasible to transit, and red nodes are infeasible to transit. With regrasping, the rotation angle is realized to advance from \(\phi_{\text{start}}\) to \(\phi_{\text{goal}}\).](image)
Grasping a driver

Grasping a hook

Attach the hook

C. Self-extension by Attaching the Hook by PR2

When PR2 wants to have a lot of things, the only two hands are not enough to realize that. So we let PR2 to use a bag the same as we put it on our shoulder. PR2 started attaching the hook whose pose is calculated with self CAD data with a driver on his shoulder in order to put a bag on his shoulder. PR2 finished attaching the hook, and the people put a lot of cans in a tote bag and put it on PR2’s shoulder. As shown in Fig. 15, PR2 realized using a bag like us human.

VII. Conclusion

This paper dealt with self-repair and self-extension by tightening screws of self-body. In conclusion, we propose below ideas and methods.

- We proposed an idea of self-repair and self-extension system by tightening self-screws for humanoid robots.
- We proposed a method of calculating the precise screw pose with self CAD data.
- We proposed a method of judging feasible regrasping of a driver in order to solve inverse kinematics of the small closed links when tightening a screw.
- We proposed a method of generating tightening motion with graph search considering regrasping a driver.

In order to verify the methods, we did some experiments of self-repair and self-extension with a real humanoid robot. As a future work, to make this system more general, we want to establish the unified method of perceiving loose screws. Also managing the tendency of loose screws is important. When the robot tightens screws of self-body, the robot memorizes it. By keeping storing the information, the robot knows which screws tend to loosen and which not. In addition, the error of grasping pose when grasping the driver is so high because the actual grasping pose and the reference grasping pose is often different, which we ignored in this paper. So the method to recalculate the grasping pose by observing the pose of the driver to the robot’s hand after grasping the driver is needed. The tightening system including those methods will help humanoid robots.

REFERENCES