Coordinate frames and differential kinematics: a survival guide

Wyatt Newman

Introduction: This set of notes introduces coordinate frames, transforms and the manipulator Jacobian. It is an abbreviated "survival" introduction. A more thorough treatment can be found in many texts on robotics. These notes are intended to help provide an introduction to robot terminology and differential kinematics to begin programming coordinated motion control on the Atlas robot (see http://www.bostondynamics.com/robot_Atlas.html) via ROS (Robot Operating System—see www.ros.org).

A robot consists of a set of controllable "joints" that allow relative motion between "links." Joints are either revolute (like a hinge) or linear (like a track or sled). This guide will consider only revolute joints (the most common type of robot construction).

Figure 1a below shows an industrial arm, the ABB IRB 120 robot. This arm includes 6 movable revolute joints, equivalent to a "waist" (a turret rotation), a "shoulder" joint, an "elbow" joint and three "wrist" joints.

Figure 1a: ABB IRB 120 robot arm

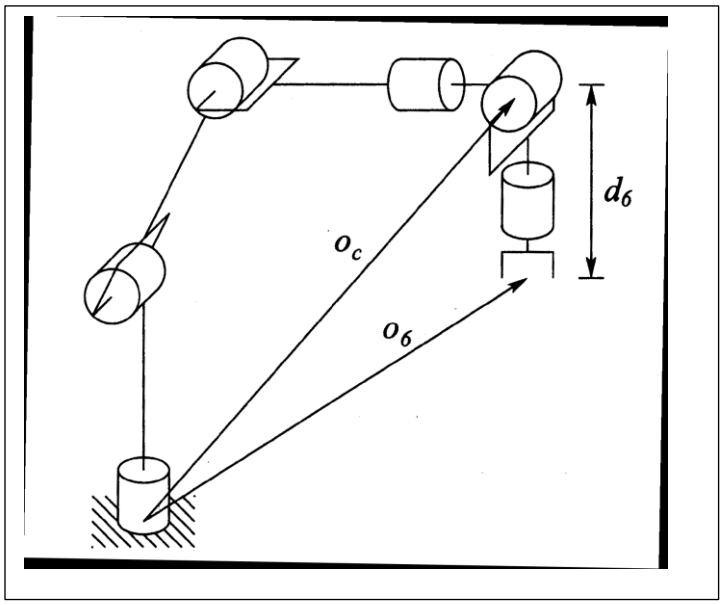

Figure 1b: kinematic abstraction of IRB 120 arm

Figure 1b illustrates a kinematic abstraction of the IRB 120 arm. In this representation, revolute joints are illustrated schematically as cylinders, and links are represented schematically as lines connecting joints. To compute the position and orientation of an end effector (e.g. a hand), it is sufficient to consider a set of constants defining the geometric relationships among the rigid components plus the angles of the movable joints.

At the lowest level, a computer controls each joint individually. Each joint has a joint sensor measuring angular rotation, and each joint has an actuator that can exert a torque. For a robot with a collection of N joints, its controller commands torques to each joint motor individually. The most common control algorithm used is "PID" (proportional plus integral-error plus derivative feedback), which maps joint states (measurements) onto desirable compensating torques.

While design of the underlying control algorithm can have many variations, it is useful to consider the simplifying assumption that the control simply "works." That is, for kinematic analysis, it is useful to assume that if a joint is given a command to go to *q* radians, the controller will do whatever is necessary to achieve that goal (i.e., resulting in the joint sensor value converging on the commanded value). This assumption will be violated due to many factors. Gravity loading and Coulomb friction in the joints will result in persistent (steady-state) errors (unless integral-error feedback is enabled). Errors can also persist if a joint actuator is incapable of exerting enough torque to overcome loading (i.e., the joint effort saturates). Errors will also persist if the commanded joint angle is outside of the range of available motion (i.e., commands exceeding the joint limits) or if the commanded motion results in interference (e.g., collision with a wall, floor, objects in the environment, or the robot's own body). Dynamic errors between commanded angles and actual angles results from commanding high-speed motions, for which the robot is incapable of following rapidly-changing commands due to inertial effects. Nonetheless, it is often a useful approximation to assume that a robot's joint angles will converge on the commanded angles. These angles are conventionally given the symbol "*q*", numbered sequentially from the base to the tip. For the ABB IRB 120 example, these joints are q_1 (the waist joint) through q_6 (the tool-flange rotation). Conveniently, the set of a robot's joint angles are stacked into a single column vector, $\mathbf{q}$.

For the Atlas robot (Figures 2a and 2b, below), there are 28 degrees of freedom (all revolute joints). Relative to the torso, each arm and each leg is conceptually similar to the 6-degree-of-freedom (DOF) IRB 120 robot arm.

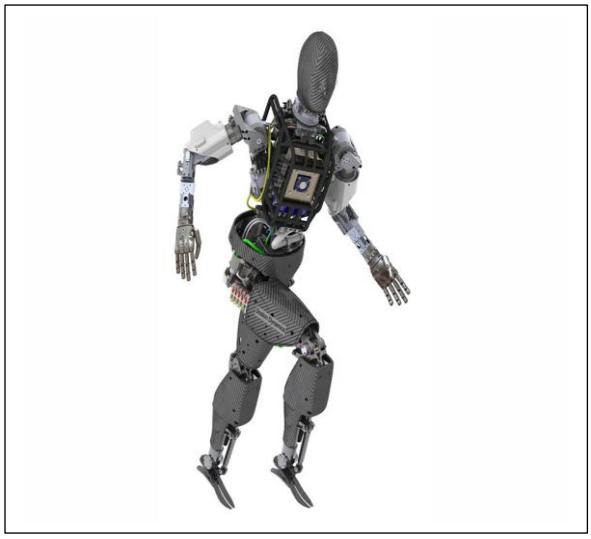

Figure 2a: Atlas robot from Boston Dynamics, Inc.

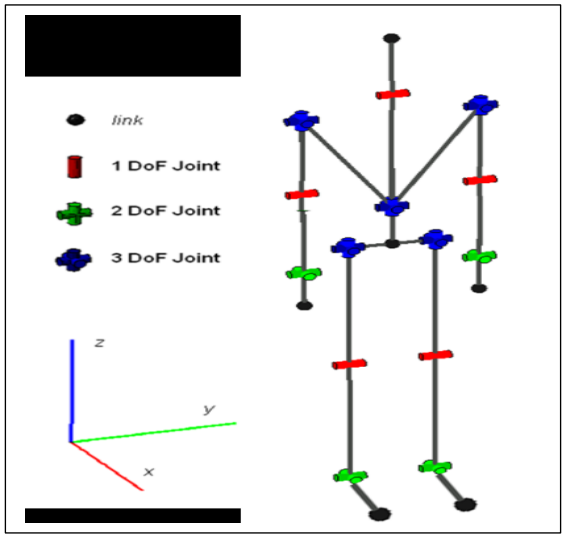

Figure 2b: Kinematic abstract of Atlas robot

A useful sub-problem in computing kinematics of the Atlas robot is similar to computing motion of the ABB IRB-120 robot. If the pelvis of the Atlas is fixed in space, and if the 3 back joints are frozen (i.e., the commanded angles of these joints are held fixed), a common question is, what is the position and orientation of a hand with respect to the pelvis (equivalent to asking what is the position and orientation of the IRB robot's tool flange with respect to its base). A complementary question is: what joint angles should be commanded in order to drive the hand to a specified, desired pose (position and orientation)?

Before we can address these questions, we must define what is meant by a pose.

Definition of frames: To compute robot kinematics (e.g., to compute the pose of a hand), it is first necessary to define precisely what is meant by a pose. The pose of a rigid body is defined with respect to a body-fixed frame and the relationship of this frame to some reference (e.g. "world") frame. Figure 3, below, shows an example of a body-fixed frame defined for an aircraft. The body-fixed frame consists of an origin as well as three directional axes. In the example of Fig 3, the body's x-axis points along the major axis of the fuselage, and the y-axis points along the wing.

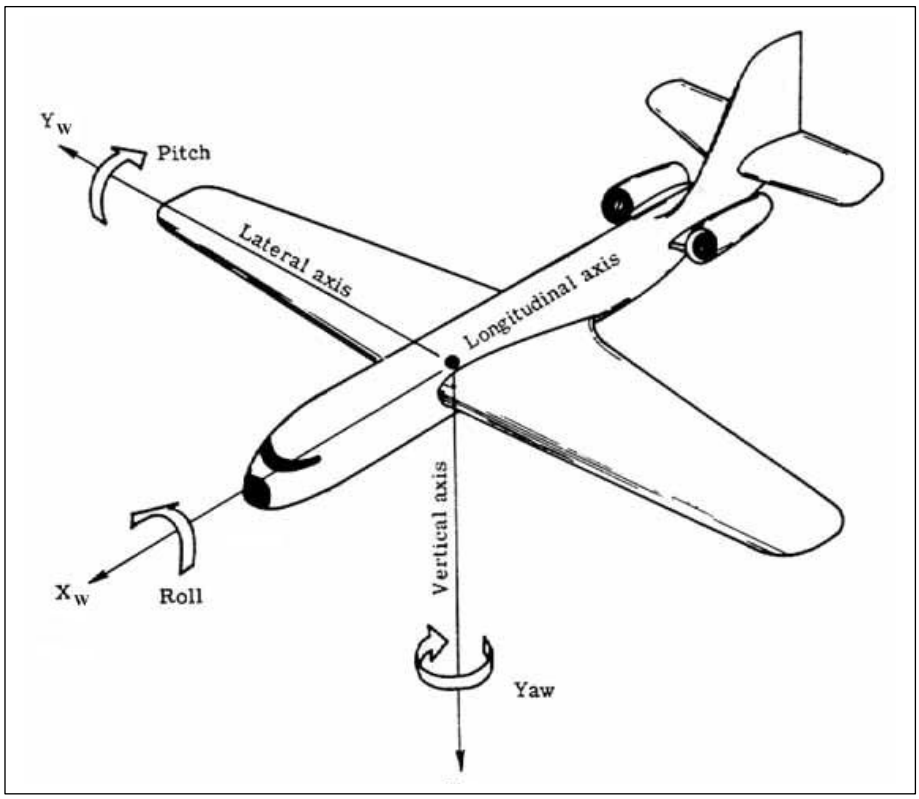

Figure 3: example of a body-fixed frame

To say "where is" this body with respect to the earth (e.g., a coordinate frame defined on a runway), we could provide numbers for the coordinates of the origin of the body's frame with respect to the reference frame by projecting the body's origin onto the reference-frame's axes, as illustrated in Fig 4.

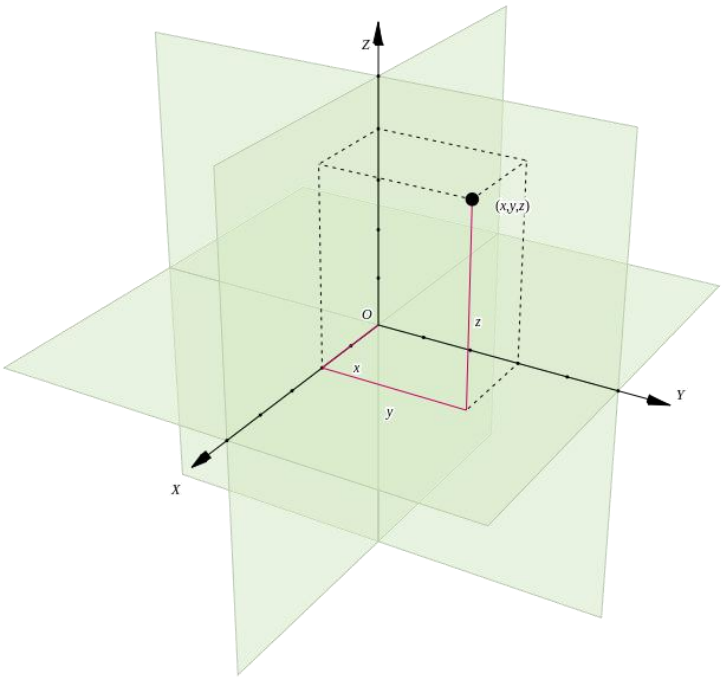

Figure 4: Coordinates of body's origin as projected onto a reference frame

As illustrated, we can define the body's origin as a point, (x,y,z)—or equivalently as a vector, $\mathbf{p}_{airplane/runway}$. This vector is equivalent to a directed line from the origin of the reference (runway) coordinate frame to the origin of by body's (airplane) frame. The notation "/" as a subscript will be used to explicitly declare what reference frame is being used.

It will be obvious from the chosen object in Fig 3 (an aircraft) that knowing the 3-D coordinates of the body's origin is not enough information. A given position of the aircrafts origin can be achieved in many ways—including level flight, flying upside down, or plummeting to earth. Clearly, specification of orientation is also important. This applies to robots as well. If we desire to grasp an object, part of the problem is to place the coordinates of (some frame origin defined on) the palm of the hand. Achieving these coordinates with the hand is necessary, but not sufficient. It is also important that the hand orientation be suitable for grasp (e.g., touching an object with the back of the hand would not allow closing the fingers about the object, as this is an unsuitable hand orientation).

Defining the orientation of an object can be confusing, and multiple representations exist, including Euler angles, quaternions and rotation matrices. Among these, rotation matrices are the easiest to visualize, and this representation will be presented here.

Although Fig 3 labels the body-fixed frame with axis labels "x" and "y" and "z", this naming choice will present confusion for coordinate transformations. The symbols x, y and z will be reserved for measurement values (scalars) with respect to a chosen reference frame. The axis labels for a body will

instead be labeled $\hat{\mathbf{n}}_i$ (for the body's "x" axis), $\hat{\mathbf{t}}_i$ (for the body's "y" axis) and $\hat{\mathbf{b}}_i$ (for the body's "z" axis). Each of these axes is a vector. In the notation used here, lower-case, bold-font letters will be used to define vectors. The "hat" symbol above a vector means that the vector is "normalized"; i.e., the vector has unit length. The subscript "i" is used to label an axis as being associated with the body named "i" (e.g. "airplane").

To describe the orientation of our object (e.g. aircraft) with respect to a reference (e.g. runway) frame, we can list the numerical values of the three frame axis vectors. To describe the direction of a vector, we need to specify values for 3 components: the x, y and z components. To do so, imagine the following process. Consider translating the origin of the aircraft to coincide with the origin of the reference frame—but do so without altering the heading (orientation) of the aircraft. Each body-fixed axis will then originate from the origin of the reference frame. We can then project coordinates of the tips (arrow-heads) of each of the three body axes onto the reference frame (identically to Fig 4), resulting in 9 numbers. For the $\hat{\mathbf{n}}_{airplane}$ axis, we can refer to the coordinates of this (translated) tip as: $n_{x,airplane/runway}$, $n_{y,airplane/runway}$, $n_{z,airplane/runway}$. (The lower-case, italic-font values are scalars). This notation is interpreted thus: $n_{x,airplane/runway}$ means the numerical value of the projection of the (translated) $\mathbf{n}$ axis of the airplane onto the x-axis of the runway frame.

Stacking the three values of the $\mathbf{n}$ axis, $n_{x,airplane/runway}$, $n_{y,airplane/runway}$, $n_{z,airplane/runway}$, defines the vector: $\hat{\mathbf{n}}_{airplane/runway}$ with three, unambiguous numerical values. Similarly, we may identify the projected coordinates of the $\mathbf{t}$ and $\mathbf{b}$ axis of the airplane to define numerical values for the components of the vectors $\hat{\mathbf{t}}_{airplane/runway}$ and $\hat{\mathbf{b}}_{airplane/runway}$.

It will be convenient to group all three vectors together in a single, 3x3 matrix as follows:

$$\mathbf{R}_{airplane/runway} = \left[\hat{\mathbf{n}}_{airplane/runway} \quad \hat{\mathbf{t}}_{airplane/runway} \quad \hat{\mathbf{b}}_{airplane/runway} \right]$$

In the present notation, an upper-case, bold-font symbol designates a matrix. While the above definition is a convenient notation for keeping track of coordinate-frame orientations, the resulting 3x3 matrix will also have valuable, alternative interpretations and uses. For the moment, however, we will simply interpret the $\mathbf{R}$ matrix as a numerical representation of an object's orientation.

We now have a complete means to specify a body's "pose," consisting of a position (a 3x1 vector from the origin of the reference frame to the origin of the body's frame) and orientation (a 3x3 matrix of components of the body's axis directions, as projected onto the axes of the reference frame). With these definitions, we can refer precisely to the pose of the airplane with respect to the runway frame by specifying numerical values for the 3x1 vector $\mathbf{p}_{airplane/runway}$ and for the 3x3 matrix $\mathbf{R}_{airplane/runway}$.

Similarly, we can refer to the pose of a frame of interest on the robot (e.g. a frame on the right hand of the robot) with respect to some reference frame (e.g., a frame associated with the pelvis of the robot).

This will allow us to precisely specify actual and desired states. For example, if we knew the position and orientation (a body-defined frame) of an object of interest (e.g., a door handle), we could define a corresponding desirable pose of a hand frame (e.g., prepared to enclose the handle with fingers). We would then have the task of moving the robot's joints such that the actual pose of the hand frame matched the desired pose of the hand frame. In general, it is a hard problem to directly compute a set of joint angles, $\mathbf{q}$, that results in achieving a target hand pose, $\mathbf{p}_{hand}$, $\mathbf{R}_{hand}$. (Note that to be complete, the reference frame must also be specified, e.g. $\mathbf{p}_{hand/pelvis}$, $\mathbf{R}_{hand/pelvis}$). This is called the "inverse kinematics" problem. On the other hand, it is straightforward to compute the complementary problem of "forward kinematics": given a set of joint angles, $\mathbf{q}$, what is the hand pose, $\mathbf{p}_{hand}$, $\mathbf{R}_{hand}$.

This computation is so common in robotics that ROS includes a standard method for computing frame coordinates (see http://www.ros.org/wiki/tf). Using this facility requires establishing a transform "listener", which may be invoked (in C++) using the ROS-defined transform listener class as follows:

```
tf::TransformListener tf_listener;  // create a transform listener
tf::StampedTransform transform;  // holder for transform result
```

A transform process computes pose coordinates, which may be accessed from the listener object. This is possible by referencing a file that describes geometric properties of the robot. Specifically, a "URDF" the file contains data that describes the relationships among links of the Atlas robot in the "Universal Robot Descriptor Format" (URDF) style (see http://www.ros.org/wiki/urdf). Each movable link gets assigned a body-fixed frame (as specified in the URDF file). The relationship between two sequential frames depends on the value of the joint angle that couples the two frames. This frame-to-frame transformation can be performed iteratively, resulting in computation of the net transformation from a frame of interest (e.g., a hand frame) with respect to a logical reference frame (e.g. the pelvis frame). This computation is already performed for you by a node in ROS, and the results are made available via member functions of a transform-listener object.

For example, with the above-defined transform listener object, one may request the transform describing the right-hand with respect to the pelvis frame with the C++ commands:

`tf::Vector3 r_hand_origin;` //establish an origin variable to be an instance of a 3x1 vector

`tf::Matrix3x3 Rmat;` //define a 3x3 matrix to hold the orientation

The following line of code populates the object "transform" with the most current available kinematic data. It should be called repetitively to recomputed the kinematics as the robot's joints move.

```
listener.lookupTransform("/pelvis","/r_hand",ros::Time(0), transform);
```

With the "transform" object refreshed with the above call, the origin coordinates of the right-hand frame with respect to the pelvis may be extracted from the result with the function:

```
r_hand_origin= transform.getOrigin();
```

The result is a 3x1 vector in "r_hand_origin" that is the coordinates of the origin of the "hand" frame as measured from the viewpoint of the "pelvis" frame.

Similarly, the orientation of the right-hand frame with respect to the pelvis frame can be extracted from the transform result with the function:

```
Rmat = transform.getBasis();
```

This operation populates the matrix Rmat with the components:

$\mathbf{R}_{r\_hand/pelvis} = \left[\hat{\mathbf{n}}_{r\_hand/pelvis} \quad \hat{\mathbf{t}}_{r\_hand/pelvis} \quad \hat{\mathbf{b}}_{r\_hand/pelvis} \right]$, which completes the specification of the hand orientation with respect to the pelvis frame.

To use the transform-listener capability, it is also necessary that the source code include the header file:

```
#include <tf/transform_listener.h>
```

It may seem odd that the above description refers to the pelvis frame. More naturally, one would like to refer to coordinates with respect to a world-defined frame. This, however, is not as simple as it might seem. In order compute the pose of the robot's hand with respect to the world, it is necessary to know where the robot is in the world. If the pose of the pelvis were known with respect to the world, then it would be a simple process to compute the pose of a hand with respect to the world (by cascading two coordinate transforms). However, knowing the pose of an object in the world is itself a challenge, called the "localization" problem. Localization requires reconciling sensor information with a model of the environment. The localization problem is easier if GPS coordinates are available—although it would still be necessary to infer the orientation of the robot (e.g. the pelvis) with respect to a world frame.

Fortunately, much can be done without knowing absolute pose coordinates in a world frame. One might, e.g., sense the location of a door handle using vision and LIDAR information. Coordinates would be detectable with respect to a sensor head. If the pose of the sensor-head frame is known with respect to the pelvis (which follows from the URDF file and from the transform listener), then the coordinates of the door handle may be computed with respect to the pelvis (a coordinate-frame transform that can be performed using the transform listener). If the goal pose is known with respect to the pelvis, and if the hand pose is known with respect to the pelvis, then these two frames are expressed with respect to a consistent reference. An objective then would be to make the hand pose (with respect to the pelvis) converge on a target pose (with respect to the pelvis). Thus, useful sensor-based operations can be performed without knowing where the robot is with respect to a world frame.

From the above argument, it is desirable to solve the sub-problem of inverse kinematics as expressed in the pelvis frame. While the inverse-kinematics problem is hard to solve explicitly, it can be addressed incrementally through the use of the manipulator Jacobian.

Differential motions and the Translational Jacobian: The "manipulator Jacobian" is a remarkably useful matrix. Abstractly, it relates incremental joint motions to incremental pose motions. In general, it

describes both perturbations of origin coordinates and orientation coordinates. In this section, we consider only perturbations of translation of an origin of interest with respect to perturbations of joint angles. The relationship may be represented algebraically as: $\mathbf{J}_{trans} = \partial \mathbf{p} / \partial \mathbf{q}$. This expression is the derivative of a 3x1 vector (e.g. origin of the right-hand frame with respect to the pelvis frame) with respect to a 9x1 vector of joint angles (6 arm joints and 3 torso joints). The result of the derivative of a vector quantity with respect to another vector quantity is a matrix. For the hand/pelvis example, the matrix of interest has 3 rows (for the 3 coordinates of the hand origin) and 9 columns (for the 9 joints that affect the hand position relative to the pelvis). In practice, this matrix may be used as:

$\Delta \mathbf{p} \simeq \mathbf{J} \Delta \mathbf{q}$. The components of $\mathbf{J}$ (27 numbers, for the hand/pelvis example) are a function of the joint angles, $\mathbf{q}$. As the arm (and/or torso) moves, the values of $\mathbf{J}$ change. For small motions, however, the incremental motion of the hand can be predicted from perturbations of $\mathbf{q}$ and from values of $\mathbf{J}$ that are assumed to remain approximately constant for the perturbation motion.

Computation of the values of $\mathbf{J}$ is less difficult than it might seem. All of the values can be computed by repeated application of computation of the influence of a single joint.

Consider the system illustrated schematically in Fig 5.

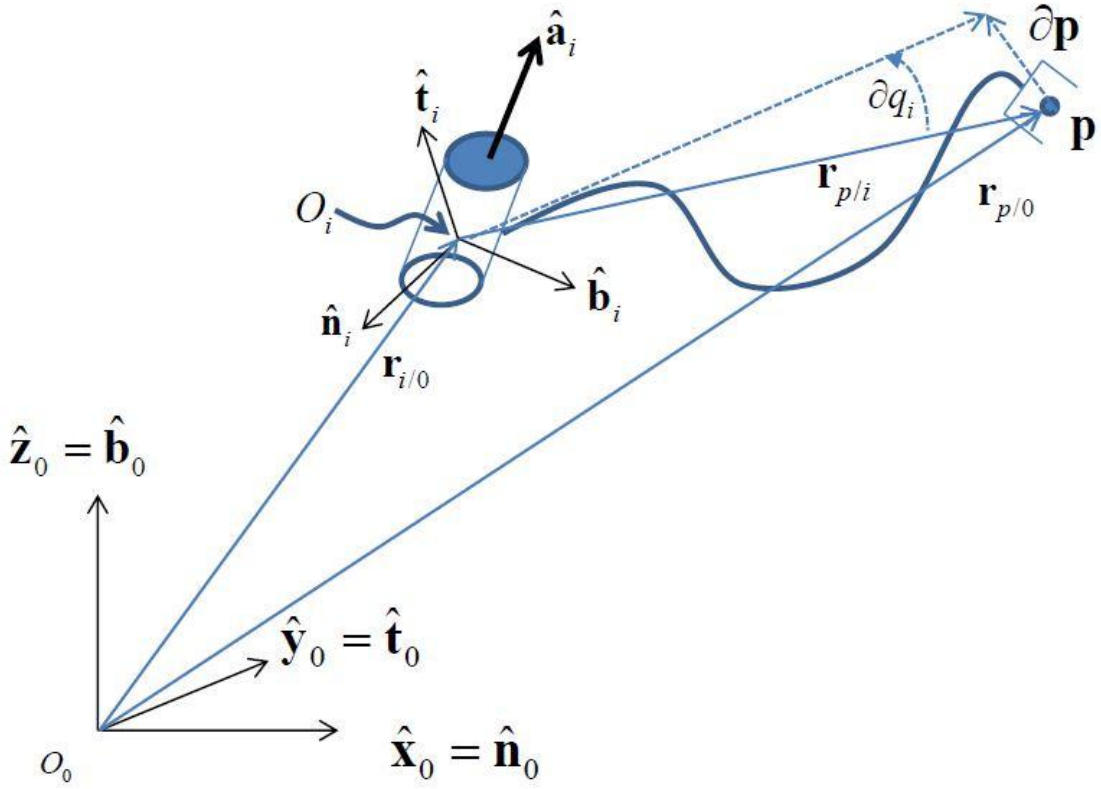

Figure 5: Illustration of Jacobian computation for joint "i"

In Fig 5, the reference frame is labeled frame "0". The frame of interest (e.g., a hand frame) has its origin at location $\mathbf{p}$. All joints of the robot are held constant, except for joint "i", which has a hypothetical perturbation of angle ∂q_i. As a result of this single perturbation, the location of point $\mathbf{p}$ moves by (vector) amount $\partial\mathbf{p}$. The motion $\partial\mathbf{p}$ results from an incremental rotation about the i'th joint axis, which has orientation $\hat{\mathbf{a}}_i$. The vector from the origin of the 0 (reference) frame to point $\mathbf{p}$ is the vector $\mathbf{r}_{p/0}$. There is also a frame associated with the link attached to joint "i" that moves as a result of motion ∂q_i. The vector from the origin of the reference frame to the origin of this i'th frame is $\mathbf{r}_{i/0}$. Both of these quantities are available via the transform listener. The vector $\mathbf{r}_{p,i/0}$ is a vector from the origin of frame "i" to the point $\mathbf{p}$. This vector could be expressed with respect to frame "i", and the components could be obtained using the transform listener. However, for the present analysis, we desire this vector to be expressed from the viewpoint (coordinate system) of the base frame (0-frame). Thus, it is computed as the difference of two vectors: $\mathbf{r}_{p,i/0} = \mathbf{r}_{p/0} - \mathbf{r}_{i/0}$.

The value of $\partial\mathbf{p}$, expressed in 0-frame coordinates, follows from the cross product:

$\partial \mathbf{p}_{/0} = (\hat{\mathbf{a}}_i \partial q_i) \times \mathbf{r}_{p,i/0}$.

Equivalently, in terms of velocities, $\dot{\mathbf{p}}_{/0} = \boldsymbol{\omega} \times \mathbf{r}_{p,i/0}$, where the "dot" notation implies differentiation with respect to time: $\dot{\mathbf{p}} = \frac{d}{dt}\mathbf{p}$, and $\boldsymbol{\omega}$ is the angular velocity, $\boldsymbol{\omega} = \hat{\mathbf{a}}_i \dot{q}_i$.

Since ∂q_i is a scalar, we can also say: $\partial \mathbf{p}_{/0} = (\hat{\mathbf{a}}_i \times \mathbf{r}_{p,i/0}) \partial q_i$

We can assign a name to the result of the cross product: $\mathbf{j}_i \equiv \hat{\mathbf{a}}_i \times \mathbf{r}_{p,i/0}$, where $\mathbf{j}_i$ is a 3x1 vector.

Note that we can get the value of the joint-axis vector, $\hat{\mathbf{a}}_i$, from the transform listener and from the URDF file. If the URDF defines the joint axis to align with one of the i'th frame axes ($\hat{\mathbf{n}}_i$, $\hat{\mathbf{t}}_i$ or $\hat{\mathbf{b}}_i$), then the value of $\hat{\mathbf{a}}_i$ can be extracted directly from the corresponding column of the $\mathbf{R}_{i/0}$ matrix (available from the transform listener). If $\hat{\mathbf{a}}_i$ is defined to be skewed with respect to the local coordinate frame, then an additional operation is required to get the components of $\hat{\mathbf{a}}_{i/0}$, expressed in the 0-frame.

From the available quantities, $\mathbf{r}_{p/0}$, $\mathbf{r}_{i/0}$ and $\hat{\mathbf{a}}_{i/0}$, we can compute the vector $\mathbf{j}_{i/0} = \hat{\mathbf{a}}_{i/0} \times \mathbf{r}_{p,i/0}$, which defines $\partial \mathbf{p}_{/0}$ proportional to the joint-angle perturbation ∂q_i.

In the above analysis, joint "i" was considered without concern for which joint this was. The logic applies to any one of the joints. For small-angle perturbations, we can approximate the net perturbation of $\partial \mathbf{p}_{/0}$ to be the linear superposition of the influences of each of the joints taken one at a time:

$\partial \mathbf{p}_{/0} \simeq \mathbf{j}_1 dq_1 + \mathbf{j}_2 dq_2 + \cdots + \mathbf{j}_i dq_i + \cdots + \mathbf{j}_m dq_m$, where there are "m" joints affecting the position $\mathbf{p}$.

We may write the above equation in matrix-vector form by stacking the j-vectors side-by-side in a matrix as: $\mathbf{J}_{trans} \equiv \left[\mathbf{j}_1, \mathbf{j}_2, \cdots, \mathbf{j}_i, \cdots, \mathbf{j}_m\right]$. This is the translational manipulator Jacobian. It relates a perturbation of a vector of joint values, $\partial \mathbf{q}$, to a vector of Cartesian coordinates of point P expressed in 0-frame coordinates, $\partial \mathbf{p}_{/0}$: $\partial \mathbf{p}_{/0} = \mathbf{J}_{trans} \partial \mathbf{q}$. For "m" joints (where m=9 for the case of hand origin with respect to pelvis), the translational Jacobian is a 3xm matrix.

Using the Jacobian to solve inverse kinematics: We mau desire frame "m" (e.g., a hand frame) to have its origin at location $\mathbf{p}_{des/0}$. From forward kinematics (and via the transform listener), we may have that the hand frame is actually at position $\mathbf{p}_{act/0}$. We desire the hand to move incrementally closer to the goal, i.e. $\partial \mathbf{p}_{/0} = \varepsilon(\mathbf{p}_{des/0} - \mathbf{p}_{act/0})$, where ε is a small scalar and the vector $\mathbf{p}_{des/0} - \mathbf{p}_{act/0}$ points in the direction from the hand to the goal. If the Jacobian were square (and invertible), then we could try to

use a set of joint-angle perturbations given by: $\partial\mathbf{q} = \varepsilon\mathbf{J}^{-1}(\mathbf{p}_{des/0} - \mathbf{p}_{act/0})$. For the example of hand motion with respect to the Atlas pelvis, the Jacobian is 3x9, and thus it is not invertible. Another option is to use the pseudo-inverse: $\partial\mathbf{q} = \varepsilon\mathbf{J}^{\dagger}(\mathbf{p}_{des/0} - \mathbf{p}_{act/0})$, where $\mathbf{J}^{\dagger} \equiv (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$, which is typically computable, even for a 3x9 Jacobian. Finally, another approach is to try $\partial\mathbf{q} = \varepsilon\mathbf{J}^{T}(\mathbf{p}_{des/0} - \mathbf{p}_{act/0})$. Using the transpose of the Jacobian is simpler to implement and is computationally faster than using the inverse or the pseudo-inverse, and this approach often yields acceptable results.

Next Steps: The preceding derivation has only considered the translational components of the Jacobian and inverse kinematics. An orientational Jacobian can also be computed, which relates differential motions of joint angles to differential changes in orientation. This Jacobian can be used to help drive a frame of interest towards a desired orientation using joint-angle perturbations, analogous to the use of the translational Jacobian.

It should also be appreciated that, to achieve satisfaction of 3 coordinates (driving a hand origin to a desired position) using more than 3 joints (9, in the Atlas hand/pelvis example) is not uniquely defined. There is an infinity of solutions (if the desired position is within reach), e.g. at different torso angles and different elbow orbits. This is a problem known as “redundant kinematics”, which involves a variety of complications and opportunities, not explored here.

The present notes, while quite cursory, offer sufficient information to provide a start for designing algorithms for intentional, goal-oriented trajectories from the Atlas robot.