1. Calibration Techniques:

A. Intrinsic
B. Extrinsic
C. ArUco Marker
D. Lidar & Camera together

A. Intrinsic

1. Data Preprocessing:

A. BAG FILES

A bag is a file format in ROS for storing ROS message data. Bags -- so named because of their .bag extension -- have an important role in ROS, and a variety of tools have been written to allow you to store, process, analyze, and visualize them.
Bags are typically created by a tool like rosbag, which subscribe to one or more ROS topics, and store the serialized message data in a file as it is received. These bag files can also be played back in ROS to the same topics they were recorded from or even remapped to new topics.

We can use the following command to get the details of a bag file:

rosbag info foo.bag

The output might look something like this:

```
$ rosbag info foo.bag
path:        foo.bag
version:     2.0
duration:    1.2s
start:       Jun 17 2010 14:24:58.83 (1276809898.83)
end:         Jun 17 2010 14:25:00.01 (1276809900.01)
size:        14.2 KB
messages:    119
compression: none [1/1 chunks]
types:       geometry_msgs/Point [4a842b65f413084dc2b10fb484ea7f17]
topics:      /points   119 msgs @ 100.0 Hz : geometry_msgs/Point
```

Fig. 1

B. IMAGE EXTRACTION

The images contained in the bag file can be extracted using the below given code snippet:
Command to run the below code:

python program_name.py bag_file.bag path_name image_topic

bag_file: name of the bag file
path_name: path of the directory where images will be stored.
image_topic: topic of the image e.g. /frontNear/left/image_raw/

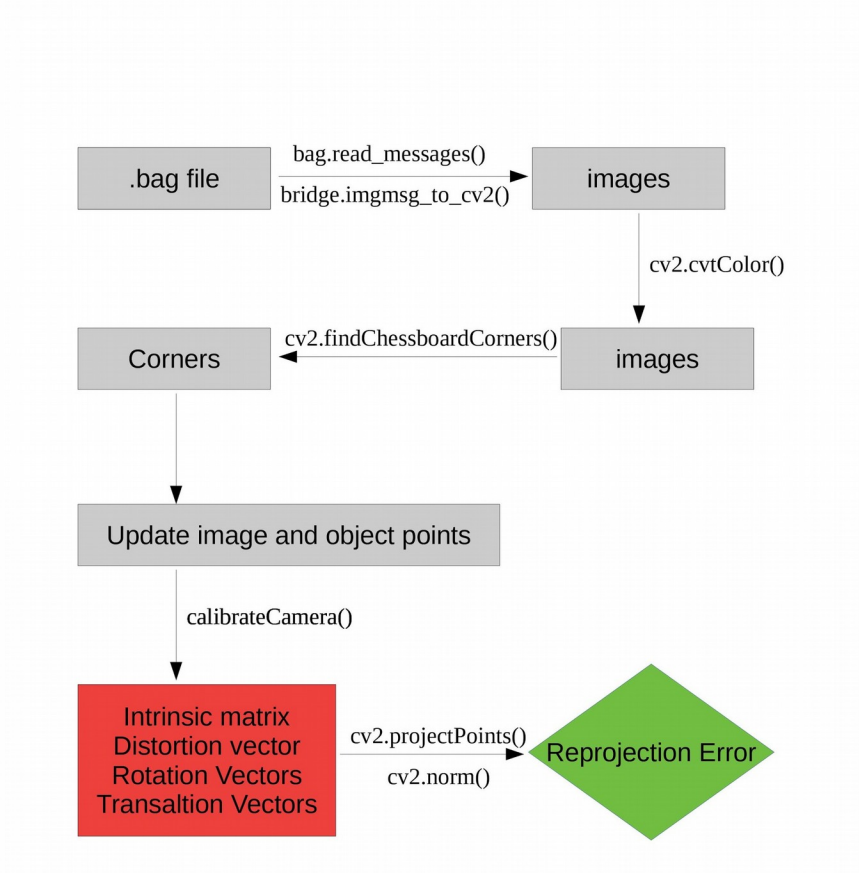
```
"""Extract a folder of images from a rosbag.
"""
parser = argparse.ArgumentParser(description="Extract images from a ROS bag.")
parser.add_argument("bag_file", help="Input ROS bag.")
parser.add_argument("output_dir", help="Output directory.")
parser.add_argument("image_topic", help="Image topic.")

args = parser.parse_args()

print ("Extract images from %s on topic %s into %s" % (args.bag_file,
                                                       args.image_topic, args.output_dir))

bag = rosbag.Bag(args.bag_file, "r")
bridge = CvBridge()
count = 0
for topic, msg, t in bag.read_messages(topics=[args.image_topic]):

    cv_img = bridge.imgmsg_to_cv2(msg, desired_encoding="passthrough")

    cv2.imwrite(os.path.join(args.output_dir, "frame%06i.png" % count), cv_img)
    print ("Wrote image %i" % count)

    count += 1

bag.close()
```

Fig. 2: To extract the images out of a given bag file

Run the above code and store the corresponding images in the left and right directories(based on the topics of the messages).
Note that the directory for storing the left and right images needs to be created before running the code.

2. Intrinsic Parameters:

A. FLOWCHART



B. IMPORTANT FUNCTIONS USED

***compare_ssim*:** Compute the mean structural similarity index between two images.

***cv2.findChessboardCorners*:** Finds positions of the internal corners of the chessboard. Returns a boolean value depending on if the given shape of corners are found, also returns location of the corners if true.

***cv2.cornerSubPix*:** The function iterates to find the sub-pixel accurate location of corners or radial saddle points.

***cv2.drawChessboardCorners*:** Renders the detected chessboard corners.

***cv2.calibrateCamera*:** Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern. Returns the rotation and translation matrix corresponding to each pattern. Also returns the camera intrinsic and distortion vector corresponding to lowest reprojection error.

***cv2.projectPoints*:** Projects 3D points to an image. Used for calculating the reprojection error.

***cv2.norm*:** Calculates the absolute or relative difference norm between the given image matrices.

C. RESULTS

The algorithm takes 40 patterns from the given sample to calibrate the given camera. To cover all the possible orientations of the checkerboard, the selected sample images must be different in orientations. Three techniques have been used to select such samples-

1. Simply selecting first 40 images: Very low reprojection error suggests a possibility of images with similar orientations of the board been selected for calibration. Therefore, a technique to select "good" images from the sample is required.

   For right images:

```
11:14:31:intrinsic$python serial.py 2019-07-19_09-53-25.bag /home/intern/devyash/intrinsic/pics1/right /frontNear/right/image_raw
Extract images from 2019-07-19_09-53-25.bag on topic /frontNear/right/image_raw into /home/intern/devyash/intrinsic/pics1/right
mean error:  0.0409229240756
camera matrix:
[[1.07570909e+03 0.00000000e+00 9.81671425e+02]
 [0.00000000e+00 1.08469867e+03 6.64583073e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients:
[[ 1.88392347e-03 -8.92942796e-01 -4.71880505e-04 -2.01482963e-03
   2.49197414e+00]]
```

   For left images:

```
11:15:50:intrinsic$python serial.py 2019-07-19_09-53-25.bag /home/intern/devyash/intrinsic/pics1/left /frontNear/left/image_raw
Extract images from 2019-07-19_09-53-25.bag on topic /frontNear/left/image_raw into /home/intern/devyash/intrinsic/pics1/left
mean error:  0.0456456227015
camera matrix:
[[1.58023707e+03 0.00000000e+00 1.05380073e+03]
 [0.00000000e+00 1.58427418e+03 5.15167149e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients:
[[-4.62603393e-01  1.04637451e+01  1.13803560e-02 -4.91027978e-03
  -1.18067129e+02]]
```

2. Skipping a few images after each image: Another approach we came up with is to skip about 20 to 25 images after selecting one so as to keep the variation amongst the selected images. The results obtained on including every 30th image are:

```
mean error:  0.3130522506
camera matrix:
[[1.23951866e+03 0.00000000e+00 9.93685000e+02]
 [0.00000000e+00 1.25137130e+03 7.12506178e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients:
[[-0.19136388 -0.00608069  0.00326984 -0.00220358  0.02222092]]
intern@intern-OptiPlex-9020:~/bagfiles$
```

3. Calculating Structural Similarity index and filtering: Involves calculating the structural similarity index to compare every image with the previous image selected for calibration, based on the assumption that each image is more likely to be similar to the image previous to it as compared to

others. A threshold value of SSIM is selected arbitrarily and the images with their value higher than the threshold are rejected. The results, obviously, are highly sensitive to the threshold value.

a. Results with threshold SSIM = 0.65

```
('mean error: ', 0.1972383949909386)
camera matrix:
[[1.33275824e+03 0.00000000e+00 1.03351932e+03]
 [0.00000000e+00 1.33872231e+03 6.29732476e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients:
[[-0.21168989  0.191585    0.0043432   0.00124483 -0.39088212]]
intern@intern-OptiPlex-9020:~/bagfiles$
```

b. Results with threshold SSIM = 0.62

```
('mean error: ', 0.2440377330627257)
camera matrix:
[[1.29390687e+03 0.00000000e+00 1.03730453e+03]
 [0.00000000e+00 1.30198262e+03 6.56381089e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients:
[[-0.23952069  0.25931037  0.00309806  0.00178206 -0.39077858]]
intern@intern-OptiPlex-9020:~/bagfiles$
```